FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

A remote monitoring and control system for ecosystem replication experiments

João Pinto dos Santos Ventura

MSc DISSERTATION

Supervisor: Nuno Alexandre Cruz

Co-supervisor: Fernando Lima

July 28, 2016

Resumo

O estudo de ecossistemas naturais é uma tarefa complicada, principalmente devido aos seus custos elevados e logística complexa. Assim, de forma a contornar essas dificuldades, diversos investigadores têm vindo a realizar experiências num sistema fechado, de forma a replicar os ecossistemas em condições controladas e facilmente replicadas e repetidas. De forma a replicar os ecossistemas mencionados, tem-se assistido ao uso de aparelhos electrónicos, tais como sensores e actuadores, de forma a controlar os parâmetros ambientais, ao mesmo tempo que é feita a monitorização do seu impacto.

Devido à globalização, que facilita a inclusão de membros de todo o mundo em equipas de investigação, existe uma crescente necessidade de controlar e monitorizar os aparelhos utilizados para a replicação das experiências de forma remota.

Actualmente, os investigadores do CIBIO têm vindo a desenvolver o seu próprio sistema para replicação de ecossistemas, que faz uso de um microcontrolador, capaz de supervisionar os sensores e actuadores utilizados. No entanto, este sistema é controlado localmente, o que obriga à presença física e portanto, de forma a possibilitar este requisito, é apresentado nesta dissertação o desenvolvimento de um sistema para controlo e monitorização remotos para múltiplas experiências, independentes entre si. Primeiro são apresentados os principais conceitos que servem de base à arquitetura do sistema, de seguida, o design final da arquitectura e, por fim, a sua implementação.

A arquitetura de sistema proposta está dividida em três camadas, sendo que a de base é composta pelos modulos de sensorização e atuação já existentes, que irão comunicar, através de um protocolo de comunicação sem fios, com um coordenador, pertencente à camada intermédia, que serve como elo de ligação ao Sistema de Gestão de Base de Dados e à Interface do sistema, que se situam na camada final e cujo acesso é feito através da Internet.

De forma a implementar esta arquitetura, é usado o Protocolo IEEE 802.15.4 para Comunicações Sem Fios, uma BeagleBone Black como coordenador do hardware associado às experiências, que é composto por um Arduino Mega, do Sistema de Gestão de Base de Dados PostgreSQL e uma Interface para o Sistema, construída com recurso a PHP e HTML.

Por fim, a praticabilidade do sistema é analisada e confirmada, sendo esta feita com recurso a vários testes onde a Interface do Sistema é utilizada para controlar os aparelhos responsáveis pelas experiências. A observação dos resultados obtidos através dos testes é encorajadora, uma vez que o funcionamento do sistema como um todo é validado, no entanto, existe ainda margem para melhorias no mesmo.

Abstract

The study of natural ecosystems is a hard task, due to its costs. and logistical complexity. So, in order to deal with these difficulties, researchers have been using enclosed experiments, in order to replicate ecosystems in controlled and repeatable conditions. To replicate said ecosystems, researchers have been employing the use of electronic devices, such as sensors, and actuators, in order to control the ambient parameters in the ecosystem, while monitoring its impact.

Due to globalization, research teams often include members from all around the world, and to facilitate the data validation, the need for remote monitoring and control of the replication experiments is rising.

Currently, researchers at CIBIO have been developing their own system for ecosystem replication, by having a micro controller supervising the sensors and actuators. Yet, this system cannot be controlled remotely, and so, in order to enable said behaviour this dissertation presents the development of a remote monitoring and control for multiple and independent experiments, namely, ecosystem replication experiments. First, the main concepts behind the system architecture are presented, secondly its design, and finally its implementation is presented.

The proposed system architecture is divided in three tiers, with the current Sensor and Actuator Nodes on the bottom tier, connected via a Wireless Communications Protocol to a Central Coordinator Node, in the middle tier, which is responsible for establishing the bridge between the Sensor and Actuator Nodes and Top Tier, which is composed by the Relational Database Management System, where the information is stored, and the System Interface, where the user interacts with the experiments.

The implementation of said architecture makes use of IEEE 802.15.4 Standard for Wireless Communications, a BeagleBone Black as a coordinator for the experiment devices, an Arduino Mega as the monitoring and control device for each experiment, a PostreSQL Relational Database Management System, and a System Interface built resorting to PHP and HTML.

Afterwards, the practicability of the system is analysed and confirmed by resorting to tests where the System Interface is used to effectively control the experiments' devices. The observation of the tests' results is encouraging, since it demonstrates that the system works as planned, although there is still margin for improvements.

Agradecimentos

Pese embora a escrita deste documento ser feita em inglês, decidi escrever estes agradecimentos em Português.

Assim, gostaria de começar por agradecer ao meu orientador, o Professor Nuno Alexandre Cruz, pela disponibilidade que mostrou ao longo do projecto e pelo facto de me ter proporcionado a oportunidade de atacar este desafio com relativa liberdade de escolha em relação à solução a implementar. Sei que fruto disso irei terminar este percurso com garantias de vir a ser um engenheiro capaz no futuro.

De seguida, gostaria de agradecer aos investigadores do CIBIO, Fernando Lima e Rui Seabra, pelo facto de terem proposto este desafio e pelo entusiamo demonstrado.

Por outro lado, gostaria também de agradecer à minha família e aos meus amigos mais próximos, por todos os momentos que passamos, saibam que sem vocês isto não era a mesma coisa.

João Ventura

"Or maybe there are no good people,
maybe there are only good decisions"

John Reese in Person Of Interest

Contents

List of Figures

List of Tables

Abbreviations

BBB	BeagleBone Black
CCN	Central Coordinator Node
DBMS	Database Management System
FFD	Full-function device
GPIO	General Purpose Input/Output
HTML	HyperText Markup Language
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
LAN	Local Area Network
LED	Light Emitting Diode
MAC	Medium Access Control
NCS	Networked Control System
PAN	Personal Area Network
PHP	Hypertext Pre-processor
PHY	Physical layer
RDBMS	Relational Database Management System
RFD	Reduced-function device
RTC	Real-Time Clock
SAN	Sensor and Actuator Node
SQL	Structured Query Language
WLAN	Wireless Local Area Network
WPAN	Wireless Personal Area Network
WSAN	Wireless Sensor and Actuator Network
WSN	Wireless Sensor Network

Chapter 1

Introduction

In this chapter the context and motivation behind this dissertation are presented, as well as the objectives proposed and the document's structure.

1.1 Context

Natural Ecosystems are biological environments that can be found in nature and ranging from aquatic to terrestrial. In the case of this dissertation, the main focus of study are aquatic ecosystems such as marine ecosystems, with high salinity and fresh water ecosystems, with low salinity levels. However, there are other aquatic ecosystems, which cannot be labeled in such as strict way. The study of the aquatic ecosystems is significant in order to determine, among others, the effects of climate change on species' distribution [1].

Aforesaid studies can be done in the ecosystem itself or through a replication of it in a laboratory. However, the study of natural ecosystems, such as the aquatic one, has been proven to be a difficult task, mainly because of the logistical issues it imposes, which are often impossible to deal with and its high cost as referred on [2].

In order to deal with these difficulties, enclosed experiments have been used, as they enable the replication of ecosystems in controlled and repeatable conditions. These replication experiments can be done in various scales concerning space, varying from a small scale, often done in small containers, medium scale, which requires the use of mesocosms, or a large-scale experiment, in which the whole ecosystem is simulated [3].

In long-term experiments, the time-span can vary from weeks to years [1], and so there is a bigger necessity for a remote monitoring and control system, since the impact of a single event, such as deviation on water temperature, can render the experiment unsuccessful. Considering that this task relies on electronic equipment, the effects of water on it cannot be neglected, as it might be the reason for malfunction or permanent damage.

In addition, said system will maximize the possibilities for an investigator to tackle other issues, without having to actively take care of ongoing experiments. Also, multidisciplinary teams, whose members might be from all around the globe, often do the conduction of such experiments.

This adds a problem, since some members will not be able to access the experiment, making them dependent on other members to view the data and control the equipment. Furthermore, the necessity for validation from peers further highlights the importance of remote access to the data acquired from such experiments, for quicker evaluation and validation of the results.

1.2 Motivation

This experiment supervising and control system is to be implemented in a Portuguese research center, CIBIO, which main purpose is centered on biodiversity and genetic resources. For further study of aquatic ecosystems, a system capable of reproducing such ecosystems is already in place. Taking into consideration the wide variety of aquatic ecosystems, there is the need for a flexible data acquisition and control system, capable of adapting to different requirements needed for each specific experiment.

Currently, the monitoring and control hardware in place is a prototype built with low-cost components, whose development has evolved through various iterations. The data logger development is reported on a paper published in a scientific journal [1], allowing anyone to freely reproduce it. However, there is a main issue, which is the inability for someone to access the data acquired remotely. The solving of this issue is critical, since deviations in various variables, such as temperature, or a power loss of the system itself, which result in the failure to replicate a specific ecosystem scenario, will render the experiment a failure, or at best, a partial success. If data was to be made available remotely, with an alarm system in place immediately informing the researchers, it would enable the possibility of saving the ongoing experiment. In the current situation, a failure of any kind will, most probably, be detected too late for any corrective measures to be taken.

Considering these issues, covering all the steps associated with systems engineering will help in achieving a system that is both accurate and reliable, meeting all the required specifications, which is the main motivation behind this disseratation.

1.3 Objectives

The main objective established for this dissertation is the implementation of hardware, such as a gateway, to connect the experiments to the Internet, enabling their remote monitoring and control. For this, wireless protocols between the gateway and the experiments are needed, since they allow physical separation. This leads to the development of a web application, with the goal of enabling data analysis and control of the ongoing experiments, remotely.

For this web application different authentication levels need to be implemented, with different access to data. The authentication level with highest clearance will enable the user to view data from the different experiments, upload new experiment settings and control the actuators. The lowest clearance authentication level must only enable the user to view the data imported from the experiments.

Furthermore, the web application must have alarms that take into consideration the data acquired in real time, in order to send notifications to the users, if the predefined thresholds are met.

Finally, after the completion of the dissertation, a paper will be written, aiming at being selected for OCEANS'16 MTS/IEEE Monterey Conference.

1.4 Document Structure

This document is divided in seven chapters, with the remainder of Chapter 1 covering the document structure.

Then, Chapter 2 presents the State of The Art on the topics related to this dissertation, mainly, Wireless Sensor Networs, Wireless Sensor and Actuator Networks, Networked Controlled Systems, Network Topologies, Wireless Communication Protocols, Single board Computers, Database Management Systems, and web servers.

In Chapter 3, the challenge and the proposed solution are presented, with detailed explanations on the architecture developed and the software and hardware chosen to play the roles described in the architecture.

In addition, in Chapter 4 there is a detalied description on how the system was implemented, mainly the firmware developed for Sensor and Actuator Nodes, the software for the Central Coordinator Node, and the System Interface. Besides this, the implementation of the chosen Wireless Communication Protocol is described, along with the relationship between all the nodes and the Database Management System.

Proceeding to Chapter 5, it contains the testing protocol followed for the validation of the system's requirements, as well as its results.

Chapter A consists on a manual detailing the various steps that a user shall take, in order to achieve a correctly use of the system. Finally, Chapter 6 addresses the conclusions obtained from the development of the system, followed by a description of future work.

Chapter 2

State of the Art

This chapter starts by presenting an introduction on monitoring and control systems, and how these can be achieved. Then, it presents the current system at CIBIO, with the following topics focusing on the literature for possible approaches in pursuance of a remote monitoring and controlling system.

2.1 Introduction

Monitoring of environments, in the last several years, has made use of networks for distributed sensing, to fully measure the parameters of interest. These networks have evolved along the years, and wireless sensor networks have been subject of broad study as they create new possibilities for sensing, by covering larger areas, and refrain from cables. With the advance of technology, which has brought down both costs and power consumptions, wireless sensor networks have become more appealing.

The necessity for controlling environmental conditions has made the inclusion of actuators on said network a logical step, thus leading to the creation of Wireless Sensor and Actuator Networks (WSAN), having the Network Control Systems (NCS) as a base for its architecture, as it enables the control and alteration of the environment by use of, for example, heating or light sources. With the incorporation of actuators, however, the complexity of the communications in the network is increased, as the protocols will have to manage many-to-one communications with sensors when these are forwarding data, and one-to-many communications when the data needs to be sent to the actuators [4]. In order to draw the complete picture on WSANs and how its components are placed on the network, further study will be made on its architecture and how can they be connected to external networks, such as the world wide web.

The link to an external network with the capacity to monitor and control the ongoing experiments is directly related to the concept of Remote Laboratories, which aims at providing all the features that a laboratory has physically, but over the Internet, such as total control of the experiments and all its instruments

2.2 Present System at CIBIO

The system currently in place at CIBIO recreates marine ecosystems, with the intention of studying the behaviour of species under certain conditions. In order to do so, it employs a micro controller, which reads and stores values obtained from multiple sensors, such as temperature, water level and luminosity, and sends output orders to the actuators, based on a schedule that is pre-programmed, such as enabling a water pump, in order to simulate tides. Water temperature is controlled using infrared lamps and cooling fans, with both devices acting according to the desired value on the schedule. The system can be repeated, in order to conduce independent experiments simultaneously. A simple system architecture is detailed in Figure 2.2, and the current hardware is shown in Figure 2.1.

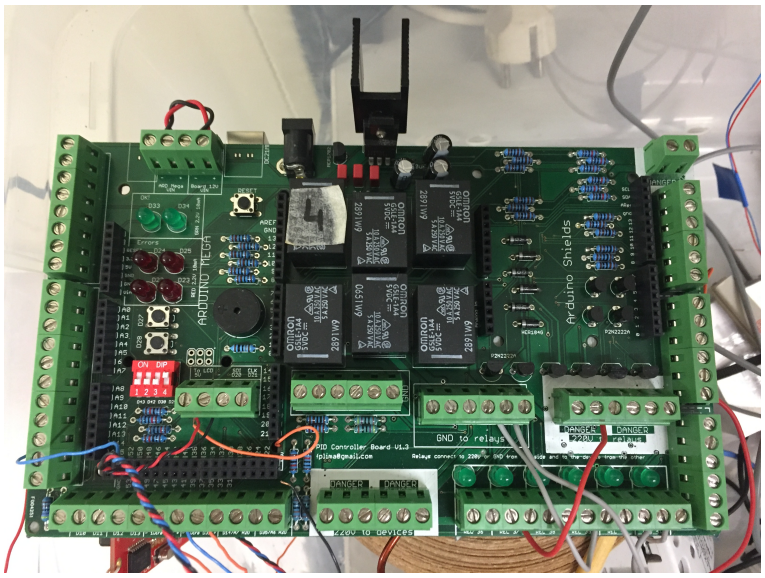


Figure 2.1: Current Hardware

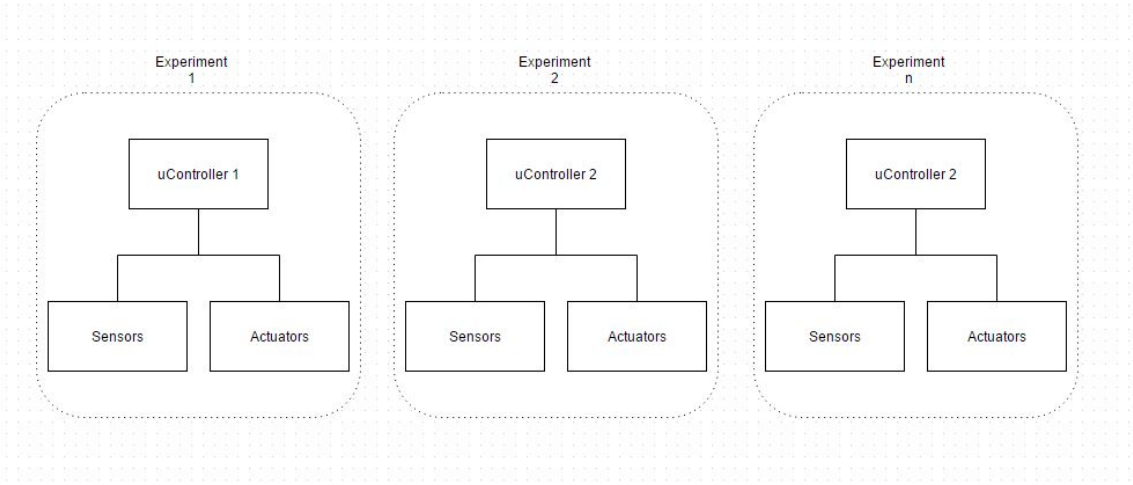


Figure 2.2: Present system architecture

Presently, there is the possibility of having underway nine experiments concurrently, however, there is no communication between them, nor to a central coordinator or supervisor. Consequently, as the experiments are independent and isolated events, with no connection to an outside network, there is no possibility of monitoring and controlling the experiments remotely at the present time.

Each experiment consists of a setup, stored on an SD Card, connected to the Arduino Mega Micro controller via a Arduino SD Shield. Said setup is constituted by a previously defined number of strings, with the same data structure, as presented on Figure 2.3, on which the user specifies the date when the actions shall take place, represented in seconds since 1970, and the remainder of the string consisting of bytes concerning the actuators' operation such as:

- Temperature, in Celsius, concerning the Set Point;
- Tide, regarding the actuation of the Water Pump used to vary the water level, with its value varying between zero and one;
- Light, which controls the infrared lights responsible for heating, also varying between zero and one, like the Tide value;

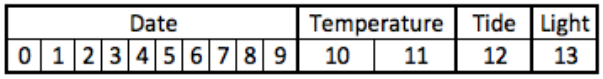


Figure 2.3: Original Setup String

The firmware in place will then load the setup from the SD card, store it on an auxiliary array. Afterwards, the array is iterated, with each of its positions containing the setup strings, by comparing the actual time, provided by a Real Time Clock (RTC), with the stored time on each string. Once the actual time is equal or superior to the time indicated on the setup string, an event is triggered, taking into consideration the Temperature, Tide, and Light Bytes.

2.3 Networked Control Systems

Networked Control Systems (NCSs) are defined by [5] as "spatially distributed systems in which the communication between sensors, actuators and controllers occurs through a shared band-limited digital communication network".

Two control systems that use network communications are defined on [6], with them being shared-controlled systems and remote control systems. On shared network connections, the sensor and actuator groups communicate with their associated controller, sharing the same network, as it can be seen on Figure 2.4.

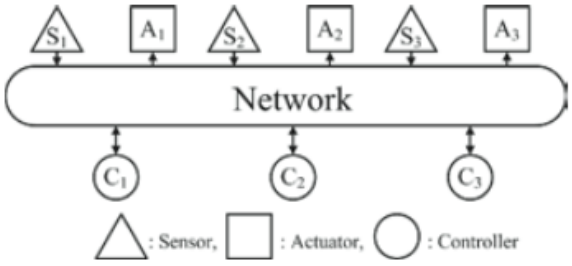


Figure 2.4: A NCS Shared Network Architecture As Presented in [6]

Remote Control Systems offers a different solution, since is has subsystems of sensor and actuators along with their controller, which then communicates with a central controller, through a network. Such architecture is depicted of Figure 2.5.

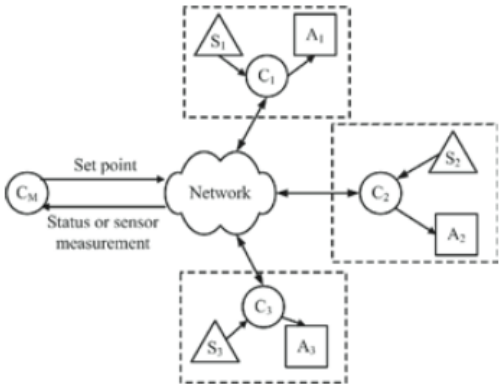


Figure 2.5: A NCS Hierarchical Structure Architecture As Presented In [6]

2.3.1 Wireless Sensor and Actuator Networks

For better understanding of Wireless Sensor and Actuators Network (WSAN), first there is the need to fully grasp the architecture and behaviour of Networked Control Systems, which were previously described, and Wireless Sensor Networks (WSN), both laying WSANs foundations.

A Wireless Sensor Network (WSN) consists on a network of sensor nodes with the purpose of monitoring a certain environment, which can either be a small, specific area, or a large area. It then forwards the data acquired to a sink (also referred as controller), using wireless communications, which can use the information or relay it to other networks, such as the Internet, through a gateway, thus enabling the interaction between user and the WSN to be made remotely. [4]

The main difference between WSN and WSAN lies on the presence of actuator nodes, which have the ability to alter the environment, either by a scheduled event or in response to an input change ([7],[4]), essentially becoming a closed-loop feedback system. In contrast to WSNs, WSANs typically require more power to operate, since it employs actuators such as water pumps or motors. Both WSNs and WSANs

2.3.1.1 WSAN Architecture

WSANs are a relatively new field of study, and as such there are still few architecture examples, however, the authors of [4] offer a possible architecture, which is presented in Figure 2.6

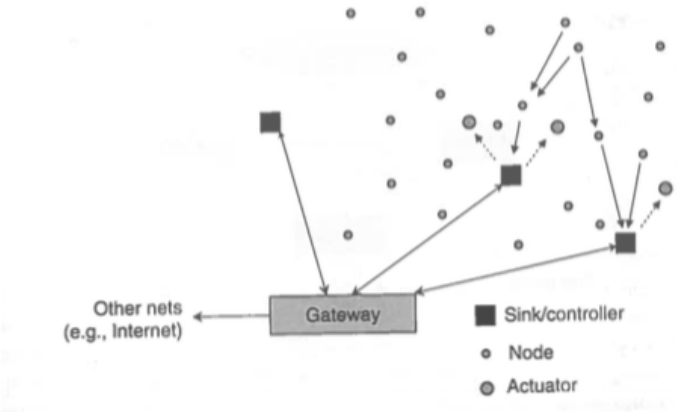


Figure 2.6: WSAN Basic Architecture As Presented In [4]

On this type of WSAN, its basic elements are nodes (sensor and actuators) connected to a local controller, that is then connected to a gateway.

Sinks are connected to the gateway, thus providing a way for connecting the WSAN to an outside network, such as the Internet. It is one of the points to be explored in this document, as it is essential for the remote monitoring and control of an experiment.

For this purpose, there is a wide range of equipment that can be used. The main focus will be on single board computer solutions, which are reliable and have the needed computing specifications in order to deal with the tasks at hand, such as establishing communications with all the sinks and relaying data between them and the user.

After further analysis on WSAN architecture, it is possible to conclude that sensor nodes, which design is presented in Figure 2.7, have the simplest hardware, since they do not need to have all the hardware necessary for actuating and controlling mechanical tools such as motors.

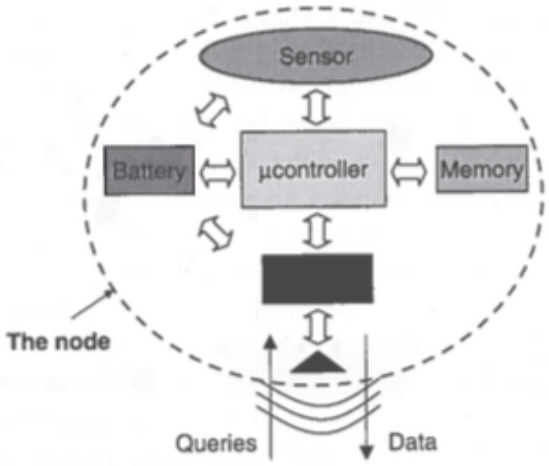


Figure 2.7: A Sensor Node As Presented In [4]

On WSANs there is also the possibility of employing integrated sensor/actuator nodes, as exemplified in Figure 2.8 which include the sensor and actuator units, a microprocessor, a battery and a means for wireless communication [8].

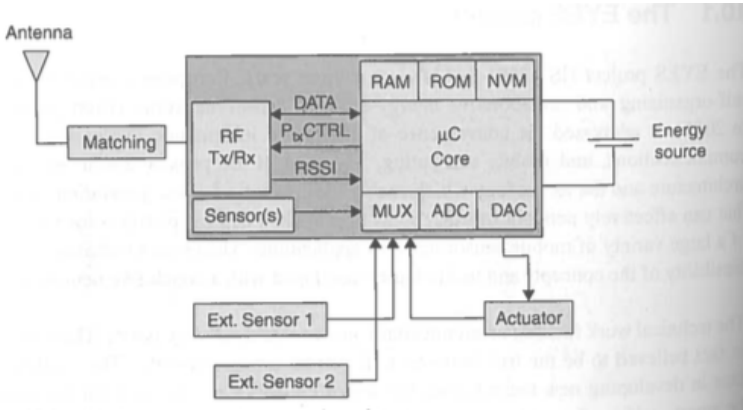


Figure 2.8: A Sensor/Actuator Node From A Case Study Presented In [4]

2.3.2 Summary

Wireless Sensor Networks and Wireless Sensor and Actuator Networks are useful tools for monitoring, or monitoring and controlling areas of various scale, from a production process, in a relatively small area, to a country's coast, or a specific area of it, as their topology permits the presence of wireless sensor nodes which can communicate with each other and correlate the data. Nevertheless, it is not an optimal solution for the required system, as even though there is the need for sensor and actuator nodes, there is no need for communication between them, as they will be assigned to independent experiments, and their spread over an area is small.

Even so, there are valuable lessons to be taken from the study of WSN and WSAN, since there is the need for a gateway in order to establish a connection to an outside network (ie Internet), which will enable to remote control the sensor/actuator nodes currently in place.

Additionally, the study of NCS also proved to be fruitful, as its possible architectures can be replicated in order to achieve a system such as the required one, since it establishes the possibility of the use of a central coordinator controller, connected to independent subsystems, as desired.

2.4 Remote Laboratories

In [9], remote laboratories are defined as online environments for operating instruments and collecting measurement data over the Internet. They are mostly used at universities for educational purposes [10], as they provide unlimited access to the experiment at all times, granting students with the possibility of autonomous work and learning, whilst preventing damages to the equipment and reducing the strain on laboratory occupation time.

In addition, [11] presents the characterization of remote experiments and how users can interact with them. In this work, experiments are classified according to the type of interaction the user can have with it, the nature of the experiment itself, and the location of both the user and the experiment. There are two types of interactions possible between the user and experiments, either by having direct control over the experiment and its equipment, or the user might control the equipment via virtual instrumentation or a virtual-reality environment.

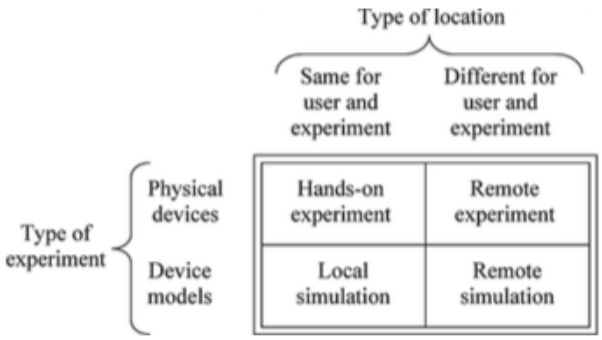


Figure 2.9: Characterization of Experiments as presented in [11]

Further along, three types of distance experiments are defined, them being based on Virtual Laboratories, Remote Laboratories with access to physical test beds and Hybrid Laboratories, which are a mix of the previous two. Although Virtual Laboratories might be of future interest, if a model of an experience is to be implemented, instead of its direct control, for this dissertation, the main point of interest lies on Remote Laboratories, and [11] sheds a light on its typical components, with them being:

- The experiment itself;
- The instrumentation devices and equipment that allow for control and results acquisition;

- The laboratory server that assures the control of said devices;
- A server links users with the Laboratory Server and Client Workstations, which assures the connection between users and the experiment alongside its resources.

Moreover, [12] sheds further light on client applications, classifying them on two groups. They can either be desktop clients running on the user's computer or web browser clients. Desktop clients have clear advantages as they provide more flexibility comparing to web browsers, however, they are more intrusive, as they have access to all the files in a computer, and its requirements are harder to meet as they might only run on specific operating systems, contrary to a web browser client, which are supposed work on every operative system with a web browser.

As web server clients have the advantage of running on almost every platform as they are not constrained by operating systems, its concept and architecture will be further explored. On [13], the authors provide insight on user classification and their interaction with the system. Concerning with the definition of user roles, three categories are defined, the web server administrator, which administrates the Web Server, the lab administrator, who is responsible for the management of lab servers as well as the experiment devices and the user, who conducts the experiment. With this information in mind, a three-level architecture is provided, such as seen on Figure 2.10.

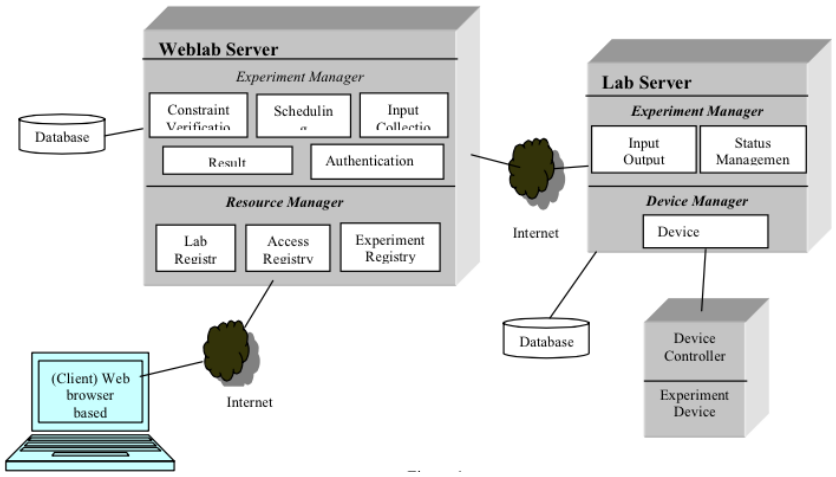


Figure 2.10: WebLab 3-level architecture as presented in[13]

Concerning this architecture, users can create and manage their account while also being able to conduct or register new experiments. The WebLab Server works both as an Experiment Manager, assuming the responsibility of authenticating user and input collection, constraint verification, scheduling and results collection, and as a Resource Manager, then acting as a creator and maintainer of lab registers, access registers and experiment registers. The lab server has tasks of experiment manager and device manager, communicating with the experiment device controller, which is the responsible for regulating the instruments used in the experiment, sending and receiving data and control signals.

2.4.1 Summary

The literature on Remote Laboratories essentially discusses the relationship between the user, the server and the experiments, with bigger focus on the first two. In essence, a Remote Laboratory is a concept similar to NCS, as both define a system that control instruments at a distance, being the Remote Laboratory specific to an application, whilst a NCS, as a concept, has broader approaches.

2.5 Embedded Devices for Remote Control

2.5.1 Single Board Computers

All the previously stated systems require hardware in order to connect the system to an outside network. Namely, in case of the NCS, the Controller, in WSANs, a gateway, or in Remote Laboratories, hardware that runs the Lab Server. In order to find the best cost/performance solution, a market analysis on open-source, single board computers was made. There are various solutions on the market, such as the Arduino Uno and Mega, Intel Galileo and Edison, Banana Pi, Raspberry Pi and Raspberry Pi 2, and the BeagleBone Black, all of which serving the purpose of a prototyping tool, and costing no more than 70€. However, the latter two will be further developed, as they are the most common solutions for embedded systems of higher complexity.

BeagleBone Black

It is a fully open source single board computer, made by BeagleBone.org Foundation that runs a Unix-based Operating System, with the same functionalities as a personal computer at a much-reduced cost. In addition to its technical features, it is also expandable, due to the existence of "capes", which are dedicated boards designed by the community, which have a wide range of applications, from prototyping to wireless communications.

Raspberry Pi 2

The Raspberry Pi 2 is a single board computer, built by the Raspberry Foundation, that like the BeagleBone Black, has the same functionalities as a personal computer, at a reduced cost. It is essentially used as a tool for education on computer based subjects and for graphical applications, but it is also used in the engineering field, either as a prototyping tool, or end-use instrument.

A comparison between the characteristics of the BeagleBone Black and Raspberry Pi 2 is show on Table 2.1.

Table 2.1: Comparison between BeagleBone Black and Raspberry Pi 2

Single Board Computers		
Characteristics	BeagleBone Black	Raspbery Pi 2
CPU Cores	2	4
CPU Clock Frequency	1 GHz	900 MHz
Memory	512 MB	1 GB
Storage	4 GB eMMC, Expandable	Micro SD Card Required
GPIO (General Purpose Input/Output)	65	40
Communication Protocols	I2C, SPI, UART	I2C, SPI, UART
USB Ports	1	4
Current Price (1st Quarter 2016)	70.05€	44.22€

2.5.2 Network Topologies and Protocols

The connection between a Gateway, Central Server, or Controller, and the experiments can be established through the use of wireless protocols, which enable to physically separate the devices, for which there are numerous topologies, such as star-shaped, mesh, ring or bus according to the possibilities of the chosen protocol, all of which will be briefly introduced.

In a star shaped topology, there is a central node that communicates independently with various nodes, rendering it impossible the direct connection between nodes. Consequently, if an end-node fails to establish communication with the central node, there is no alternative way for the flow of information to reach the central node [14].

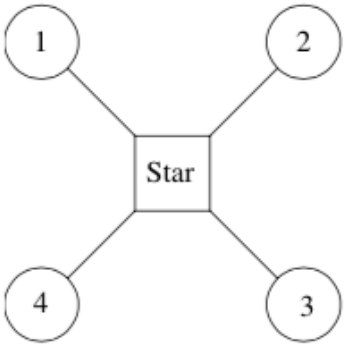


Figure 2.11: Star Topology As Presented In [14]

Moreover, in this topology, each node can listen to all the transmissions sent by the central node directly to it, akin to a bus topology in a cabled network. However, as the mean of communication between the end-nodes and the central node is wireless, the distance between them must be taken into account, since its increase might undermine the possibility of end-nodes to listen to all the information sent by the central node.

On a mesh topology, in contrast to a star topology, its end-nodes are interconnected, providing alternative routes for the flow of information, which is useful if direct connection to the central node cannot be established [14]. Nonetheless, in comparison to Star or Ring Topology, it is more complex to manage.

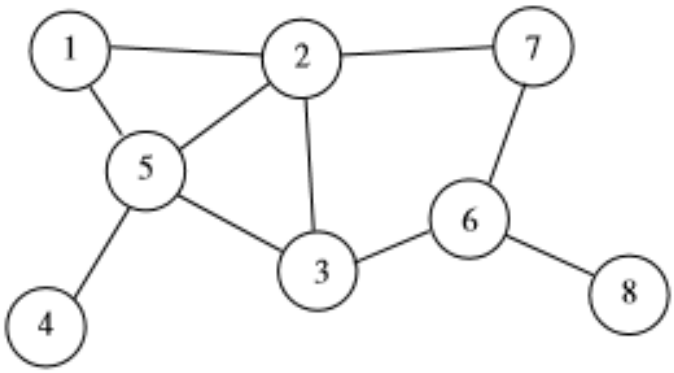


Figure 2.12: Mesh Topology As Presented In [14]

On ring topology, the main node and the end-nodes are sequentially connected in a ring, making the flow of information circular. As a result, information sent by a node will have to go through all of nodes between it and the destination. This may prove to be a major disadvantage, as the failure to communicate wit one node will cripple the network in such a way that the flow of information will be halted [14]

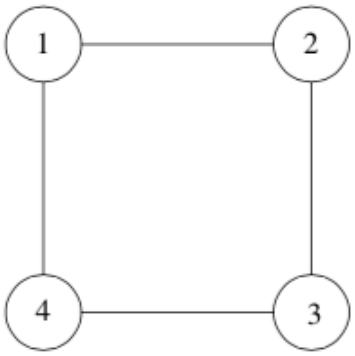


Figure 2.13: Ring Topology As Presented In [14]

These network topologies, can be implemented making use of various wireless communication protocols. As of today, the most common protocols for it are Wi-Fi (IEEE 802.14), ZigBee (IEEE 802.15.4) and Bluetooth (IEEE 802.15.1).

Wi-Fi

The Wi-Fi protocol, short for Wireless Fidelity, is defined by IEEE 802.14 standard for WLANs, allowing access to the Internet when connected to an Access Point, or in ad-hoc mode[15]. The

main purpose of this protocol is to replace or extend wired networks responsible for computer-to-computer communications.

ZigBee

ZigBee Alliance Company created ZigBee standard in 2002. Its protocol stack is built upon IEEE 802.15.4 that specifies the PHY and MAC layers. It is a self-configuring, self-healing system of redundant and low-power nodes [16], highly interesting for the automation industry and, most specifically for WSNs, in consequence of its low cost and low power consumptions [17], which is a critical requirement in this type of networks.

The implementation of devices in this protocol are divided in two types concerning their functionality: Full-function device, which has 3 operation modes: Personal Area Network (PAN) coordinator, coordinator and device, and Reduced-function device, that can only communicate with full-function devices.

Bluetooth

Bluetooth, or IEEE 802.15.1, is a standard designed for short-range devices, based on wireless radio system applied on wireless personal area network and its topology provides two connectivity options, piconet and scatternet [15]. On a piconet, one Bluetooth device acts as the master, while one up to seven devices connected act as slaves. The communication between the master and the slaves might be point-to-point or point-to-multipoint, with the slaves synchronized by the master's clock. Scatternets consists on the connection of piconets in order to form a wider network, with the particularity that a master in a piconet can be a slave in several other piconets. One particular advantage of Bluetooth is the ability to place a device in standby mode in pursuance of power consumption reduction.

The comparison between the characteristics of the 3 Wireless Communication's Protocols is shown on Table 2.2

Table 2.2: Protocol Comparison with values adapted from [15] and [18]

	Protocols		
Characteristics	ZigBee	Wi-Fi (a/b/g/n)	Bluetooth
Range	10-100 m	250 m	10 m
Max number of nodes	>65000	2007	8
Frequency Band	2.4 GHz (Global); 868 MHz (Europe)	2.4 GHz; 5 GHz	2.4 GHz
Max Signal Rate	250 Kb/s	600 Mb/s	1 Mb/s

Summary

After comparing the three protocols, Bluetooth might be of lesser interest if the number of nodes is high (equal or more than 8). The ZigBee protocol has the maximum number across the three protocols, and it requires lower power with its signal rate being more than enough for the application in cause. Moreover, in [15] it is also stated that ZigBee offers less complexity as it presents the lower sum of both PHY and MAC primitives.

2.6 Data Storage on the Internet

The aim to store data, in a structured way, and make it accessible via the Internet, leads us to database management systems. A computerized database may be created and maintained either by a group of application programs written specifically for that task or by a database management system.

2.6.1 Databases

On Traditional database applications, the information stored is mostly textual or numeric, however, there are new types of database systems, often referred to as big data storage systems, or NoSQL systems, which have been created to manage data for social media applications [19].

According to [20], a database system typically consists of four components: users, the database application, the database management system (DBMS), and the database, as shown on Figure 2.14.

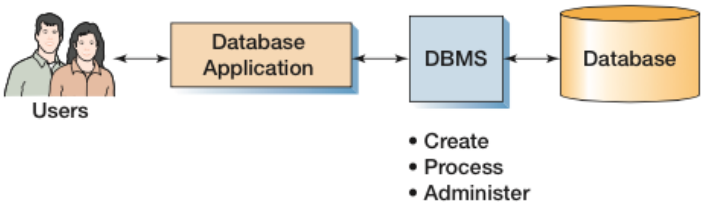


Figure 2.14: Components Of A Database System As Presented In [20]

Due to the significance of Structured Query Language (SQL), and the fact that it is understood by all DBMS products while also being usually used by database applications to send statements to the DBMS, [20] presents an alternative architecture for the database system, as shown in Figure 2.15.

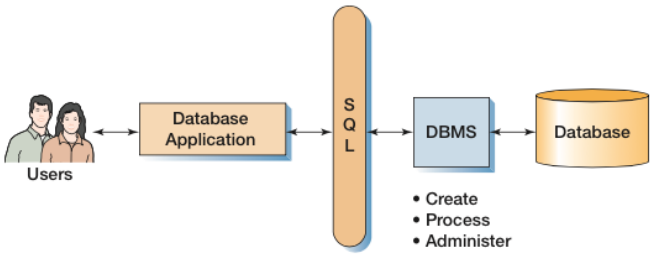


Figure 2.15: Components Of A Database System With SQL As Presented In [20]

[19] defines the Database Management System as a "computerized system that enables users to create and maintain a database, while facilitating the processes of defining, constructing, manipulating, and sharing databases among various users and applications".

The Database Application role is to serve as an intermediary between the user and the DBMS, and they can read or modify data present on the database by sending SQL statements to the DBMS.

Additionally, they might also present data in the format of forms and reports. Although application programs can be acquired from software vendors, system specific solutions might be developed and employed.

Functions of a DBMS
Create database
Create tables
Create supporting structures (e.g., indexes)
Modify (insert, update, or delete) database data
Read database data
Maintain database structures
Enforce rules
Control concurrency
Perform backup and recovery

Figure 2.16: Functionalities Of A DBMS As Presented In [20]

The evolution of DBMS led to the arrival of Relational Database Management Systems (RDBMS), which main difference to DBMS, is that besides storing information in tables, they also allow to store the relationships between them [21]. As of today, the most popular, open-source, Relational Database Management Systems are MySQL and PostgreSQL.

To build the database applications, there are several tools available, such as Hypertext Markup Language (HTML) and Hypertext Pre-processor (PHP). While HTML is useful for generating static Web pages, with fixed objects and text, when it comes to interactive features, PHP is a better solution, as is particularly suited for manipulation of text pages, and in particular for manipulating dynamic HTML pages at the Web server computer, as referred in [22].

Taking this into consideration, a three-tier architecture is reached, as shown of Figure 2.17

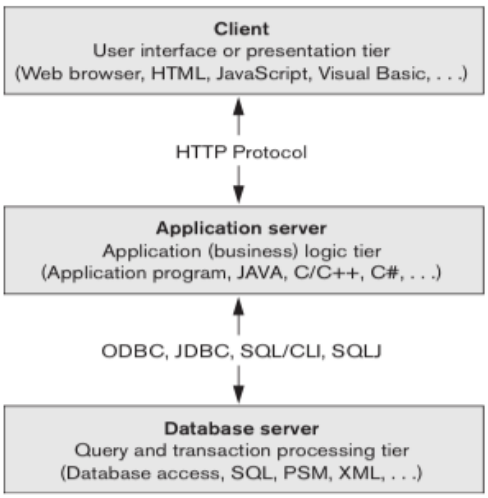


Figure 2.17: DBMS Three-Tier Architecture As Presented In [20]

2.6.2 Webservers

The term Web Server refers to hardware, software or a combination of both. When mentioning the hardware part, a web server is a computer, or another physical device, connected to the Internet, which hosts the files associated with a web site, and handles the requests made to web pages by users. Looking at Web Servers from the software perspective, a web server controls the way users access the files hosted on the hardware, in order to execute the requests made by users [23]. There are numerous software web servers, and some of the most popular are the:

- Apache HTTP Server, which is open source and runs on the likes of Linux, Mac OS X, and Windows;
- Microsoft IIS (Internet Information Services) which is runs Windows operating systems;
- Sun Java System Web server, from Oracl, which runs, among others, on Linux and Windows operating systems;

Even though there is no real way to confirm this, the Apache HTTP Server seems to be the most popular web server software, and that being the case, to host a database, developers often use a combination of Apache-MySQL-PHP. This combination, when running on the Linux operating system, is referred to as LAMP; when running on the Windows operating system, it is referred to as WAMP [22].

Chapter 3

Challenge and proposed solution

3.1 Presenting the challenge

This dissertation aims at achieving remote monitoring and control of several experiments. As described in subchapter 2.2, the system in place only allows for presential control of the experiments, and each time a user wants to start a new experiment, he has to physically access the experiments site, remove the SD Card from the SAN, and then store the new setup. Ideally, the desired final system shall be able to access each SAN, communicate with it, either by sending it orders, or logging data, and then relay it to an user on the website. In order to achieve this goal, there are system requirements to be met, which will be further explained.

3.2 System Requirements

To achieve a functional system there are several requirements to be met, some are general, related to the whole system, and some are specific to each individual block.

General Requirements:

- The system needs to handle, at least, 9 experiments at the same time.
- The experiments are all placed in the same physical space, with a few meters between them.
- The experiments maximum duration is 12 months;
- The connection between the SANs and the CCN must be wireless.
- Data has to be stored on a database;
- There should be a backup energy source for the central controller, to keep it operational in case of a power supply failure.
- A Web Site is required as the System's Interface, and must be accessible via internet;
- Notifications on unexpected system behaviour must be sent directly to the user;

In order to implement the General Requirements presented, the following guidelines were taken into consideration.

Website Guidelines: On the website, the user, after logging in, shall have the ability of performing the following actions:

- Insert new experiment setups;
- Check data from all the experiments, in real time, being able to download it;
- View real time or completed plots from chosen experiments;
- Check the state of each experiments, with warnings for lost communications, and re-established communications;
- View which setups are available, and which SAN are without setups and available to start a new experiment;
- Check the last received message from any of the SAN;
- Start a new experiments;
- Receive notifications on unexpected system behaviour via email.

3.3 System Architecture

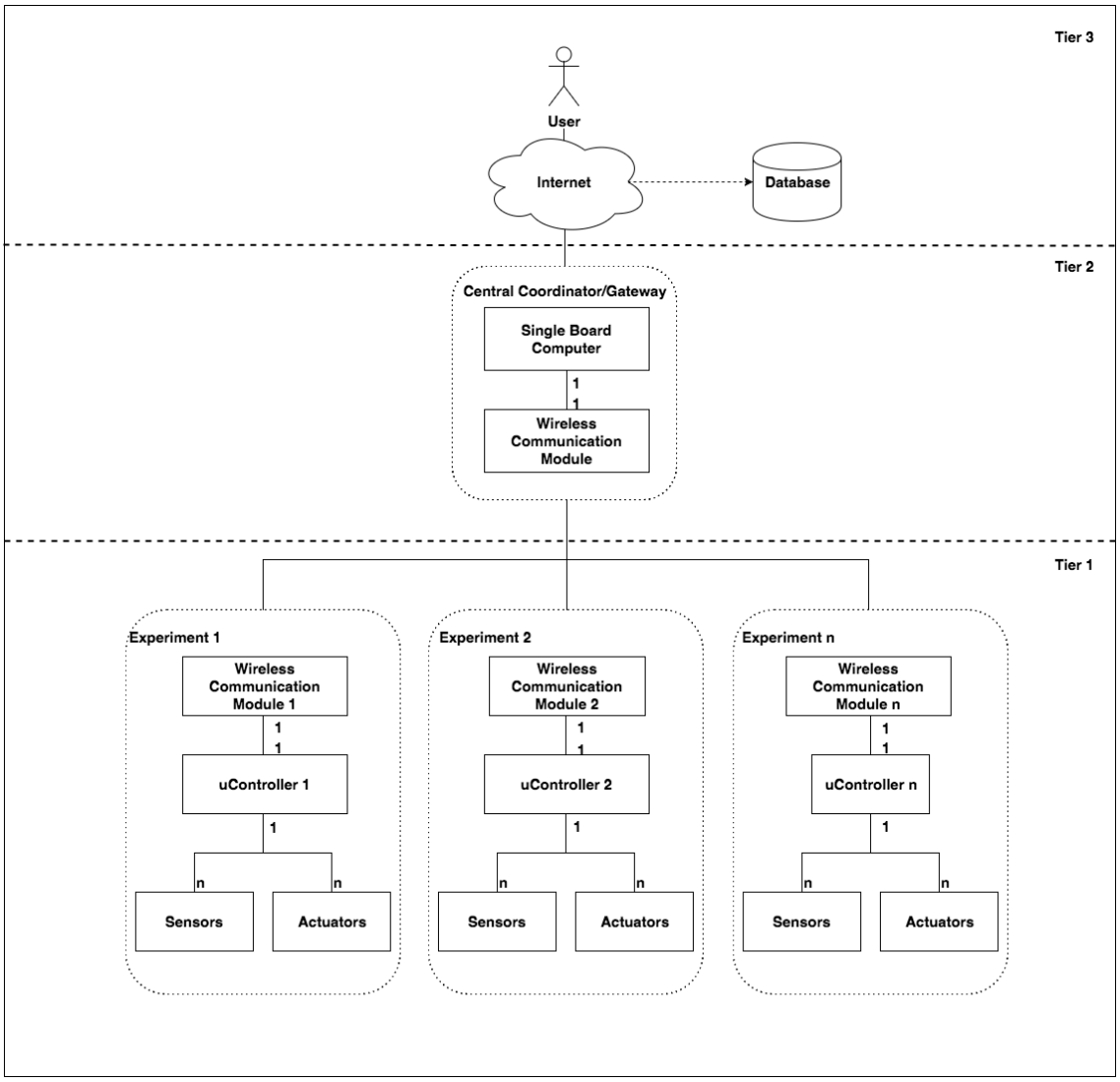


Figure 3.1: System Architecture

The main objective of this System Architecture is to integrate the present Sensor and Actuator Nodes (SANs) in a network, enabling the users to control and monitor the experiments remotely.

Within the architecture, each of the tiers have distinct purposes. Begining from the bottom tier - 1 - the SAN, on which the architecture was built upon, are responsible for controlling each experiment, by running a pre-defined setup, monitorize the experiments, and send periodic logging messages.

The middle-tier - 2 - consisting of the CCN is responsible for establishing the connection between several SANs and the database, meaning that it has to be capable of receiving messages from the SANs, process them, and act upon it, either by replying to it, or storing the information on the database.

The upper-tier - 3 - is divided in two sub-tiers. First, we have the DBMS, which has various roles in this system, such as storing the logging data from the active experiments, storing the setup data, and information about the active SAN on the network, and their current state. Then we have the System Interface, which gives the user the ability to monitor the experiments, via graphics and tables, and supervise the SAN state, with notifications of registered system malfunctions. Additionally, it must store the information regarding users, such as their username, password and email.

3.3.1 Design Criteria

The design of the architecture relayed heavily on the study made on WSN, WSAN and NCS presented in Chapter 2, as each of these architectures present us with a distributed architecture where we have sensor, or sensor and actuator nodes relaying information to a gateway. From the gateway on, the design was inspired by Remote Labs and Database Management Systems.

It shall be able to establish communications with each SAN individually, giving orders, receiving data, and relay it to a database. Given that the maximum number of nodes is limited, the option for the network architecture fell upon a Star-based design, with the CCN being the network coordinator, and each SAN connected directly to it. As previously stated, one of the system requirements was for this network to be wireless. All the data is to be primarily stored on a database.

3.4 Hardware and Software Selection

Taking into account the system requirements, and the architecture developed to deal with them, the next step was to select the tools used to implement the system.

Taking into consideration the study presented on Table 2.2, the IEEE 802.15.4 protocol for Wireless Communications was chosen, mainly because of the superior number of nodes that can be connected simultaneously, which is higher than the maximum number for either Bluetooth and Wi-Fi. Even though the Wi-Fi has superior range, that is not a critical requirement, as the experiments are separeted by a few meters.

With the network protocol chosen in mind, and given the extensive support by the community, reasonable price, and technical specifications, such as being able to establish a Star-Shaped Network, the choice for the wireless module fell upon the Xbee PRO Series 1, which has the following characteristics:

- Retries and Acknowledgements;
- Direct Sequence Spread Spectrum (DSSS);
- Source/Destination Addressing;
- Unicast And Broadcast Communications;
- Point-to-point, point-to-multipoint and peer-to-peer topologies supported
- Indoor/Urban: up to 300’ (90 m), 200’ (60m) for International variant;
- Outdoor line-of-sight: up to 1 mile (1600m), 2500’ (750 m) for International variant

To connect the Xbee Pro Series 1 to the SAN (which is an Arduino Mega), the Wireless SD Shield for Arduino is the selected hardware. The main reason behind this choice is that, being based on the Xbee Modules from Digi it can use any module with the same footprint, meaning that if, in the future, there is the need to change to a different module, the hardware will not burden that change. Furthermore, the Wireless SD Shield has an SD slot, which can be used to store or retrieve data.

Advancing to the middle-tier of the architecture, the choice for the Central Coordinator Node was the BeagleBone Black. The decision to choose the BeagleBone Black over the Raspeberry Pi 2, was made by taking into consideration its MMC, which means that it doesn't require the presence of an SD Card, even though it might be employed for memory expansion purposes. In contrast, the Raspberry Pi 2 requires the presence of an SD Card, since is does not have a MMC, meaning that the OS will run on the SD Card. This gives the BeagleBone Black an advantage, since it will be possible to store information on the SD card, regarding the Setups, as a backup to the database. This way, SD Cards might be replaced without making an impact on the normal behaviour of the BeagleBone Black. Furthermore, the BeagleBone Black has more General Purpose Input/Outputs, and even though that is not a specific requirement, it adds the possibility of future expansions, such as adding security sensors like temperature and humidity ones. Furthermore, in more complex situations, it might even be used to replace the Arduino Mega as a SAN.

To connect the Xbee Module to the BeagleBone Black, Sparkfun's XBee USB Explorer was the preferred choice, since it simplifies said connection, as it is done by connecting the XBee USB Explorer to the BBB's USB port. Besides, the Explorer can also be used to connect the Module to a Personal Computer, which is helpful when one wants to configure the module, with the help of X-CTU application.

Moving up to the third-tier, the RDBMS was employed by using PostgreSQL. The main advantage of PostgreSQL in comparison to MySQL is the better implementation of the ANSI norms. Furthermore, the familiarity with PostgreSQL was the most preponderant reason for the choice of PostgreSQL over MySQL. The System Interface will consist on a Website, where the user controls and supervises the experiments. This website was built using PHP, which is a server side scripting language, containing a wide arrange of libraries, with numerous applications. Furthermore, PHP supports DBMS - Database Managements Systems - such as MySQL and PostegreSQL, which was a fulcral requirement, as this System Interface needs to access information stored on the Database [22]. Amongst the use of PHP, HTML4 and 5 are also to be used in the construction of the website.

Chapter 4

Implementation

In this chapter, the implementation of the system architecture presented in Chapter 3.3 will be described, with emphasis on the software developed for the different tiers, and offering some further insight on the Xbee Modules.

4.1 Data Packets

The basis for the correct functioning of the SAN relies on the Setup Data, which is stored on a string, as presented on Chapter 2.2, and used to control the actuators, by following the reference values stored in it. To minimize the impact on the firmware previously developed for the SANs, the aproach taken was to include a header, containing the Setup ID and the SAN ID, as a redundancy measure, since it is then possible to compare the Setup and SAN ID received on each of the setup's string, to the previously assigned Setup and SAN ID, enabling the detection of errors, mainly the possibility of a specific SAN to receive a Setup that was allocated by the user to another SAN, to both the original setup string, shown in Figure 4.1, and log strings, shown in Figure 4.2.

Setup ID	SAN ID	Date										Temperature		Tide	Light
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Figure 4.1: Setup Data Packet

On the setup string, the header includes the Setup ID, which is selected by the User, and the SAN ID, which is given by the CCN after consulting the database. Both of these aspects will be further developed during the chapter.

Message ID	Setup ID	SAN ID	Timestamp										Temperature Reading					Temperature Setpoint		Temperature Error	Temperature Output	Water Level Reading	Water Pump State	Lights State
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

Figure 4.2: Log Data Packet

The approach taken for the log string, which contains the data acquired by the SAN was similar. Like the Setup Data Packet, it stores the values associated with Setup ID and SAN ID, but

it also stores a Message ID, which is assigned by the CCN once it receives the acquired values by the SAN. The remainder of the Log Data Packet stores the time, in seconds since 1970, when the data was acquired, and the values read from the sensors, and the actuators current state.

The way these strings are constructed is the pivotal point of the implementation, since it englobes the way setups, and log, are sent by the SAN, received and processed by the CCN, stored on the database, and displayed on the System Interface.

Both Data Packets are transmitted in Xbee Network Messages, which handle the message's error control. The Xbee Network Messages are thoroughly described later in this Chapter.

4.2 Relational Database Management System

In chapter 3.4, it is referred that the choice for a Relational Database Management System fell upon the PostgreSQL. In order to have access to the PostgreSQL RDBMS, a program called XAMPP, which runs the services associated with web page management and RDBMS was employed. To facilitate the use of the RDBMS, phpPgAdmin was used. The use of such tools is documented along this section.

The RDBMS is responsible for storing data information regarding setups, experiments logging, and SANs ID, moreover, current setup and state are stored. On top of that, the information regarding the users, such as their user name, email and password is also stored.

4.2.1 Administration Tool

phpPgAdmin is a web-based administration tool for PostgreSQL, built on php and javascript [24], that requires a host (web server), in order to work. It makes easier for users to Create Tables, and to Insert, Update, or View data on tables, due to its user-friendly nature.

4.2.2 Web server

As described in chapter 3.4, the Database and the System Interface shall be hosted outside the BeagleBone Black. During the development of the integrated solution, the best way for testing would be to have a PC hosting both the website and PostgreSQL in a Local Area Network.

4.2.2.1 XAMPP

XAMPP is a free Apache distribution, containing MariaDB, PHP and Perl [25], and was used to host a PostgreSQL RDBMS, through the Apache Web-server, which hosts HTTP. Through Virtual Hosting, the Apache Web Server provided by XAMPP is capable of hosting multiple websites simultaneously. With that in mind, it is used to host both the database and the System Interface website.

4.2.3 Entity Relationship Model

To achieve the final database design, the network architecture and the system requirements presented in Chapter 2 where taken into consideration. In the end, the following tables were created, as presented in Figure 4.3.

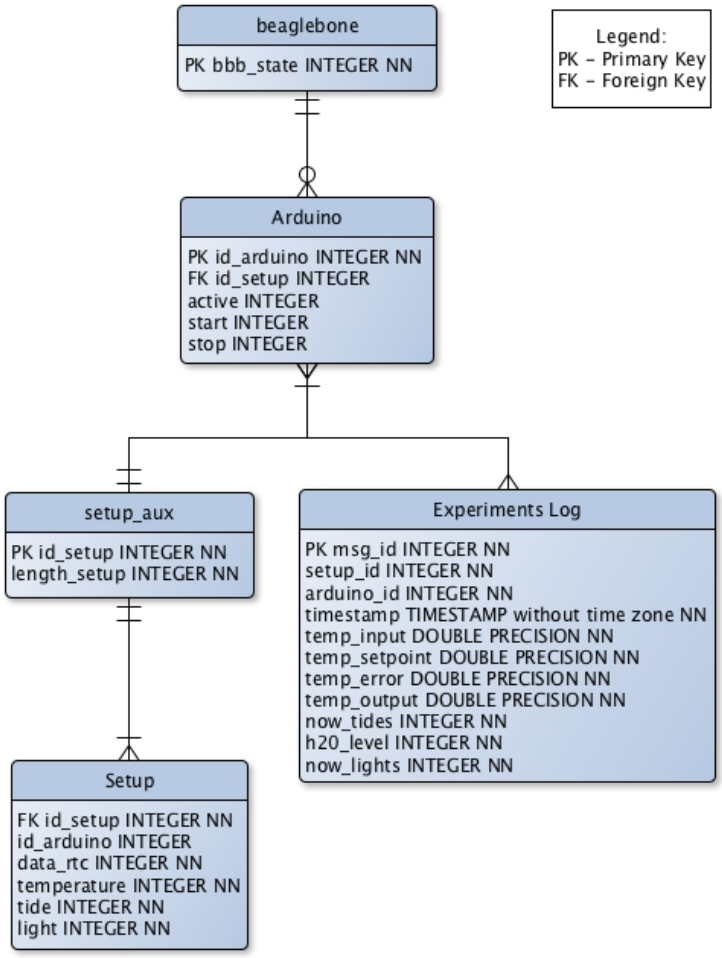


Figure 4.3: Entity Relationship model

This model was developed with these characteristics in mind:

- There is a Single CCN (beaglebone Table) connected to multiple SANs (Arduino Table);
- A SAN (id_arduino on the Arduino Table) can only have one setup at any given time;
- One setup (id_setup on the setup_aux Table) has multiple strings, with the reference values for the experiment, which are stored on the Setup Table;
- A SAN has multiple experiments log messages, which are stored on the Experiments Log Table;

When the system is up and running, on the *beaglebone* table, whenever the CCN is turned On, the "bbb state" variable is Updated to 1, and when it is turned Off, it is updated to 0.

The Arduino Table contains the information about the SAN, and one of the main concerns when implementing the database was to impede the system to give replicate SAN IDs. To solve this, the *id_arduino*, which contains the SAN ID, was created as a Primary Key, meaning that its value cannot be repeated on any other row on the table.

Once a SAN receives an ID, and sends back an acknowledge message, the active value is updated to 1. From that moment on, the User is able to start a new experiment with it. The system should be flexible enough so that several SANs can run the same setup at the same time, as the user might want to validate the experiments data, by running it, in parallel, on different SANs. When the experiment starts running the start value will be updated to 1, and once it ends, it goes back to zero, and stop value is updated to one. The main reason for the "id arduino" to be the only Not Null variable is due to the fact that, when the SAN establishes communications with the CCN and an ID is assigned, a row can be inserted in the database with only that value in it, leaving the other fields as Null, and available for future updates when required.

The Setup Aux Table stores the information regarding the Setup Identification and the number of strings that a specific setup has. Both are Not Null values, to avoid the user to, unintentionally, store setups with no identification or null length. This table is directly related to the Setup Table, where each row represents a string containing the information presented on Figure 4.1. Once more, the possibility of entering incomplete setups in the system is taken into consideration, with the solution, once again, falling on the Not Null restriction,.

The last table, Experiments Log, stores the logging data from all the SANs. Each message received comes with an ID (msg_id), given by the CCN, so that it becomes easier for the user to identify each message. The timestamp variable is a TIMESTAMP without time zone, which automatically converts the date in epoch (seconds since 1970) to date and time in a conventional way (20YY-MM-DD HH:MM:SS). The decision to apply the Not Null restriction on this table was made in order to identify if there is a problem going on with the system, since whenever the CCN tries to Insert a new row on the table, if values are missing the PostegreSQL will return an error message.

4.3 Network Configuration

In this section the different modes of operation of an Xbee PRO Series 1 are presented. Taking that into consideration, a decision on the mode of operation was made, and details are given on the configuration of the Xbee PRO Series 1 Module, considering its role in the Network that is being established.

4.3.1 Xbee PRO Series 1

The Xbee PRO Series 1 is an RF Module, operating with base on the 802.15.4 IEEE Standard For Wireless Communications. Its modes of operation and characteristics, along with the selected parameters will be explained in this subsection.

4.3.1.1 Modes of Operation

The Xbee PRO Series 1 can operate in two modes:

- Transparent Operation (AT), which is the predifined mode of operation, and enables establishment of communications between Xbees in a transparent way, meaning that no data packets have to be formed, the user simply sends the string, and the Xbee will send it to the Xbee configured as the destination. This means that whenever the destination of the message is to be changed, the user will have to use the command mode, in order to change the destination address, which is a complex approach.
- Application Programming Interface (API), which extends the level to which a host application can interact with the networking capabilities of the module, enabling users to transmit highly structured data quickly, predictably, and reliably [26], since it provides alternative ways to configure the modules and route information.

Given the complexity of the AT Mode when multiple Xbee Modules are connected, the API mode was chosen, and its data structure is presented on Figure 4.4.

Figure 4.4: Xbee API Mode Data Structure As Presented in [26]

According to [26] the API Mode offers the following advantages:

- Transmitting data to multiple destinations without entering Command Mode;
- Receive success/failure status of each transmitted RF packet;
- Identify the source address of each received packet;

Moreover, the API mode has two submodes:

- API Mode, with Figure 4.4 presenting its structure, which eliminates the sequences containing escaped characters, and differentiates API frames by taking into consideration only the start delimiter and length bytes, meaning that if there is a loss of bytes in a packet, the length count will be off, and the next API frame is also lost.
- API Mode with Escaped Characters, which structure is presented in Figure 4.5, with added reliability, since it enables Escaped Characters to be interpreted as data, instead of control characters.

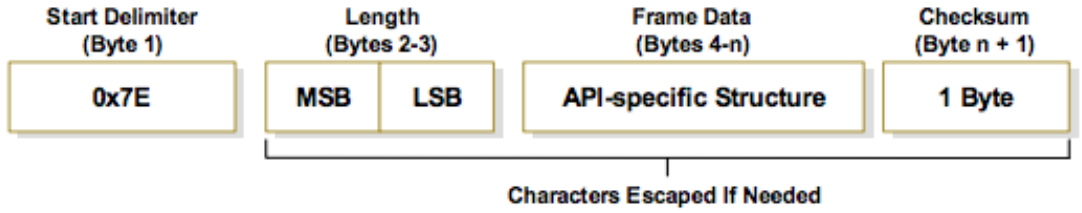


Figure 4.5: Xbee API Mode 2 Data Structure As Presented In [26]

The previously mentioned Escaped Characters are data values that must be flagged so that they will not interfere with UART communications. The characters that must be flagged are:

- 0x7E – Frame Delimiter;
- 0x7D – Escape;
- 0x11 – XON;
- 0x13 – XOFF

Besides of the characters flagged, the data frames between the two is similar. The data contained in the frame has a length ranging from 4 to n bytes, and the information that it carries depends on the action taken. Independently of the mode of operation chosen, the Xbee Modules have two address types:

- The Short Type, hexadecimal and user defined, which length is 16-bit;
- Long Type, hexadecimal, which length is 64-bit;

In order to use the Long Type Address, the 16-bit address must be disabled and this is done by defining the MY parameter as 0xFFFF or 0xFFFE. Once that is done, the module's address becomes the 64-bit IEEE address, which is stored in the SH and SL parameters.

For this project the modules are configured to work with 64-bit addresses only, and there are only two types of messages implemented Tx, shown in Figure 4.6, and Rx, shown in Figure 4.7.

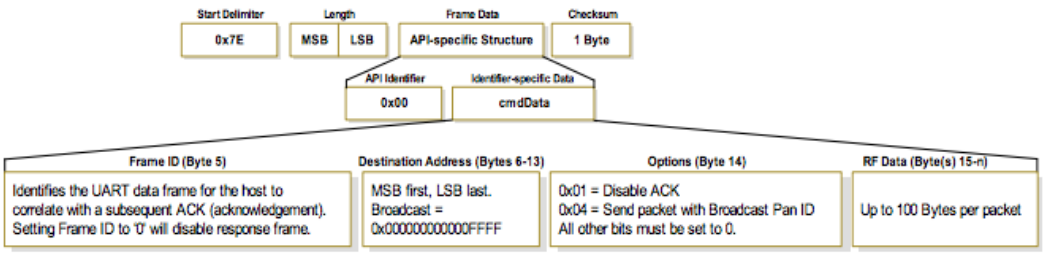


Figure 4.6: Xbee 64-bit Tx Request As Presented In [26]

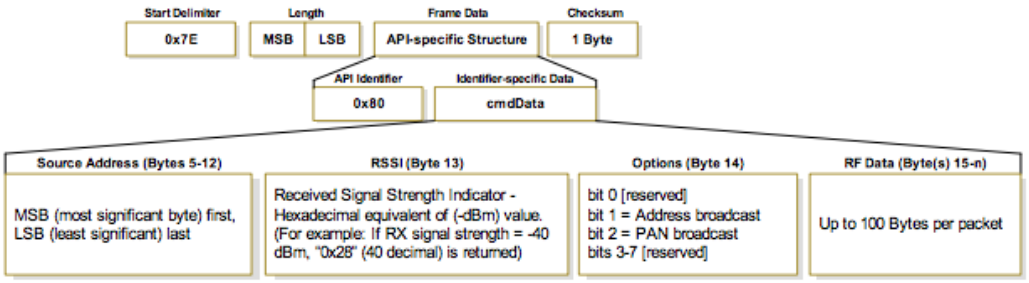


Figure 4.7: Xbee 64-bit Rx Request As Presented In [26]

Because of these choices, only the coordinator module, present on the CCN, is able to send broadcast messages (Indirect Transmissions), and End Devices, present on the SANs, can only send direct Transmissions, in this case to the CCN.

4.3.1.2 Configuring the Xbee Modules

The Xbee Modules were configured by using X-CTU, which is the official program that Digi, the entity behind the Xbee, provides, in which the user is presented with a graphical interface, presented in Figure 4.8, to manage and configurate Xbee Modules.

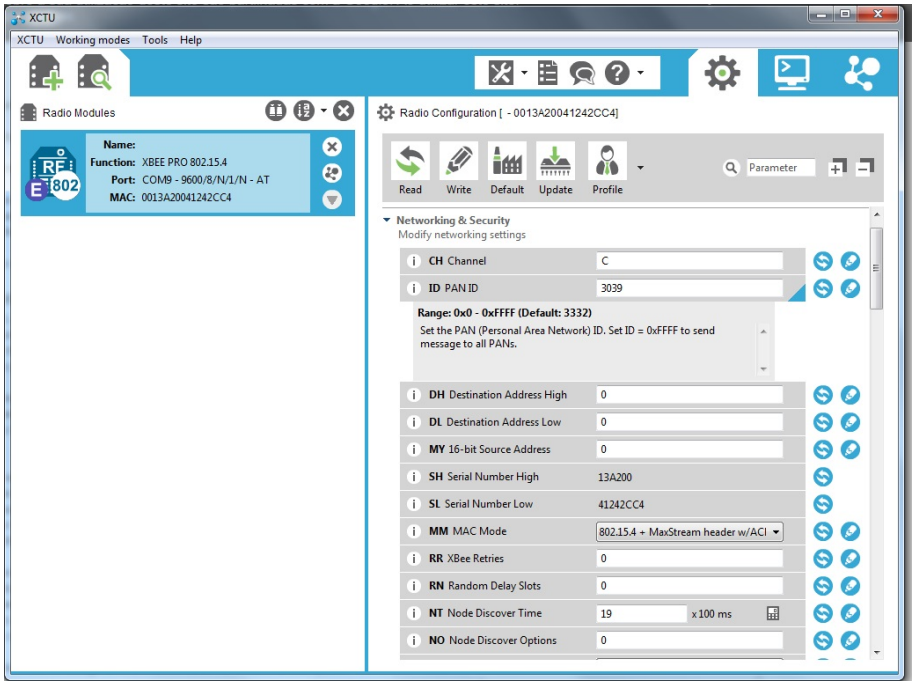


Figure 4.8: X-CTU Interface

To set up a network that enables the modules to communicate with each other, there are several steps, similar in each module, that must be followed.

The first step when configuring the Modules is to set the Channel number and the Personal Area Network (PAN) Identifier, both being the same for each module, an example of the procedure is shown on Figure 4.9.

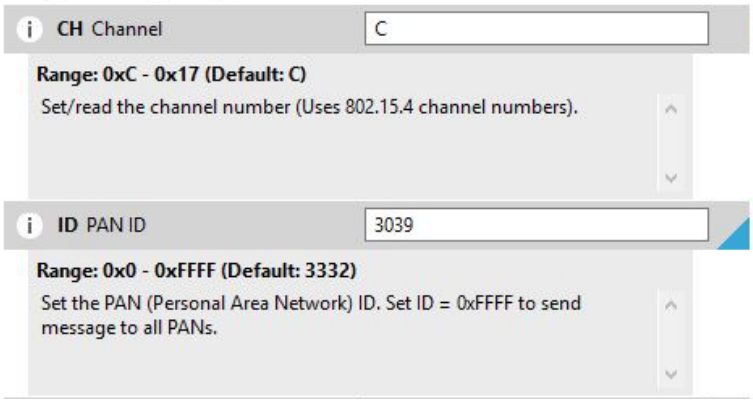


Figure 4.9: 802.15.4 Channel number And PAN Identifier

Afterwards, the baud rate is set to the chosen value, 9600, which is the default value on each of the Xbee Modules. This configuration is shown in Figure 4.10

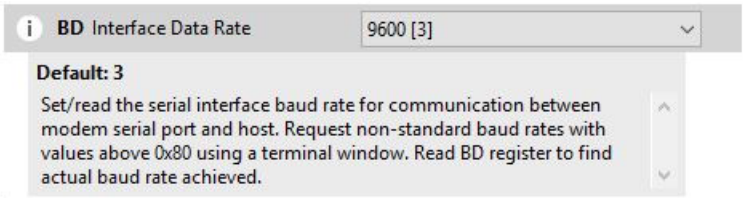


Figure 4.10: Setting the Xbee Baud Rate

Finally, one must choose the mode of operation, shown in Figure 4.11, which was, as previously explained, API Mode 2, with escaped characters.

Figure 4.11: Xbee Mode of Operation Selection

Moreover, the X-CTU also enables users to check the module's 64-bit address, as shown in Figure 4.12.

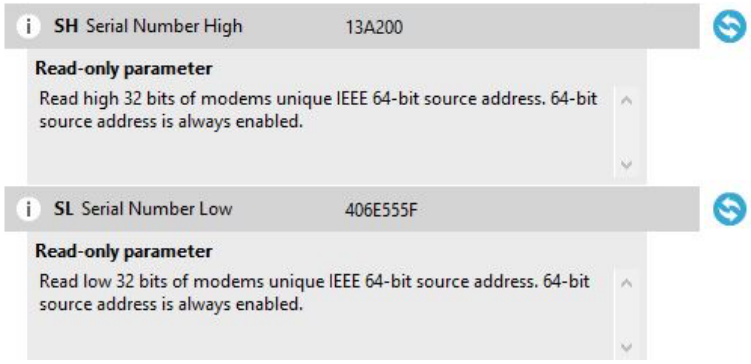


Figure 4.12: Xbee IEEE Module Address

As previously stated, there was the need to disable the 16-bit address of each Xbee Module, this was done by setting the MY Value to 0xFFFF, as shown in the Figure 4.13 .

Figure 4.13: Disabling Xbee 16-bit address

To check if the modules are working correctly, the X-CTU provides us with a terminal, which can be used to send or receive various types of messages, therefore, enabling the exchange of messages between several Xbee Modules.

4.4 Central Coordinator Node

According to the system architecture presented in chapter 3.3, the basic functioning of the CCN consists of:

- Establishing connection to the internet and the database;
- Accept communications from multiple SANs, replying immediately once a message is received, which is also described as a callback;
- Manage the SANs.

As such, this section focuses heavily on the software developed, further delving into the Integrated Development Environment (IDE) and libraries used, and the software's organization.

4.4.1 Software

4.4.1.1 Cloud 9

The IDE chosen to developed the code for the CCN was Cloud 9, which is an open-source web based programming platform that supports several programming languages, and installed on the BBB by default.

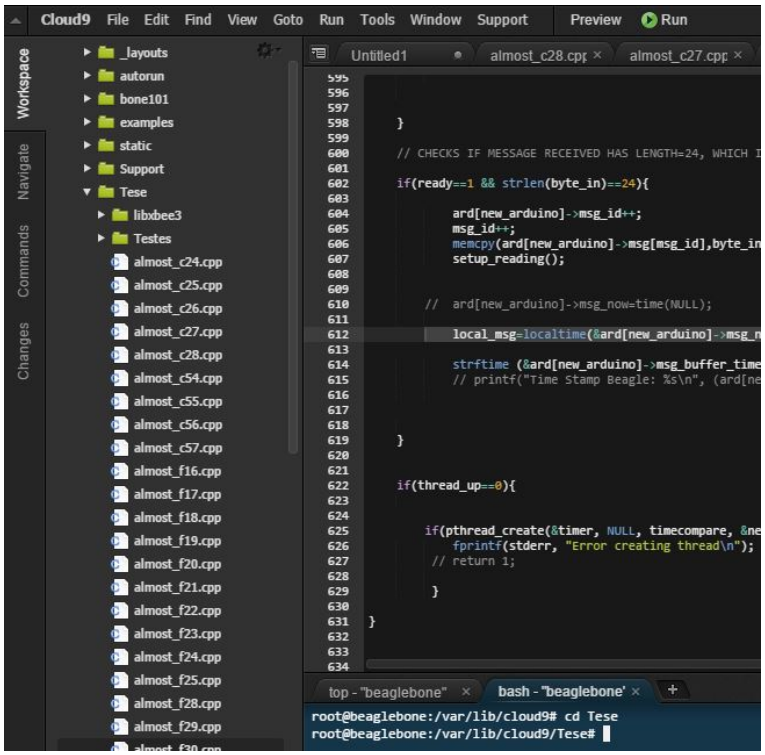


Figure 4.14: Cloud 9 Interface

4.4.1.2 libxbee

One of the main issues with the Xbee Communications in API mode, is the increasing complexity in its implementation in comparison with AT mode. In order to facilitate said implementation, there are several Libraries, some officially backed by Digi, while others are developed by the Xbee Community. Given that this project is being developed with C/C++ languages, the choice fell on libxbee. Its files are available for download on [27]

This library provides users with a simpler way to interact with the Xbee Modules in both its API submodes, making it easier to construct, receive and analyse data packets.

For the development of the software, the following functions defined on the library, and with further characterized in [28] were used:

- *xbee_conNew()* - Creates a new connection to the Xbee Module, by specifying the connection type and the module's address;
- *xbee_conSettings()* - This function is used to configure the module's settings, such as catching all packets, enable broadcasting, or disable acknowledges;
- *xbee_conCallbackSet()* - Sets a callback function associated with an established connection;
- *xbee_conCallbackGet()* - Retrieves data acquired from the callback set with a connection;
- *xbee_conTx()* - It is used to send a message to a module with an established connection;

4.4.1.3 libpq

In order to establish an interface between the CCN and the PostgreSQL, the use of libpq C library [29], which is "the C application programmer's interface to PostgreSQL" was required, since it enables the program to send queries and receive its results.

For the development of the software, the following functions defined on the library, further explained on [29], were used:

- *PQconnectdb()* - Establishes a connection with the database server;
- *PQstatus()* - Returns the state of the connection with the database server;
- *PQerrorMessage()* - Returns information on the errors;
- *PQexec()* - Sends a command to the database;
- *PQresultStatus()* - Returns a value used to check errors;
- *PQntuples()* - This function returns the number of rows returned from the query;
- *PQgetvalue()* - Returns the value from a single field on a single row;
- *PQfinish()* - Ends the connection established with the database;

4.4.2 Algorithm

For further comprehension of the algorithm implemented, it is important highlight that, in order to store all the data regarding each individual SAN a structure was developed, as presented on Listing 4.1.

```
1  typedef struct Arduinos{
2    int arduino_id; //ID Assigned to the SAN;
3    int setup_id; //Setup ID Assigned to the SAN;
4    int msg_id; //ID from the Last Logging Message Received;
5    char setup[1000][20]; //Setup, stored in an array of strings, with each position
         of the array being one of the strings pertaining to the setup;
6    char msg[1000][30]; //Log Messages Received, stored in an array of strings, with
         each position of the array being one of the strings pertaining to the Log;
7    int setup_length; //number of rows of the setup
8    int active; //Flag used to identify the SAN's communication state;
9    int start; //Flag used to
10   time_t msg_now; //Timestamp of the last logging message received;
11   double seconds_id; //counter used to check time passed between received messages;
12   char msg_buffer_timestamp; //CCN Actual Time;
13   xbee_conAddress address_arduino; //SAN 64-bit IEEE Address;
14   int interrupted; //Number of times there was a loss of communications;
15 }Arduino[10];
```

Listing 4.1: Data Structure for data received from the SANs

Once it starts, the CCN's *main()* routine, which is one of the steps presented in Figure 4.15, establishes a connection to the Internet, and then gets the time from an Network Time Protocol (NTP) server, which is essentially for synchronizing all the data, before connecting to the PostgreSQL RDBMS.

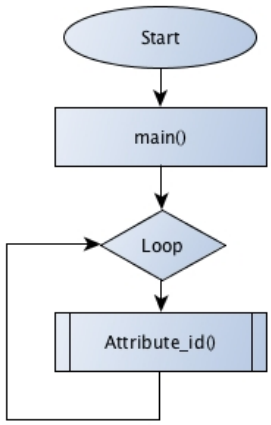


Figure 4.15: Client Sequence

After this step, the CCN opens its Xbee communications, by calling the function *attribute_id*, as illustrated in Figure 4.16, and waiting for a message from a SAN. Once a message is received, the function *xbee_CatchAllCB()* checks if the message arrived from a new or a recurring node, by comparing its data packet address with the stored ones. Each time this function is called, a new thread is opened, once the callback is ended, the thread is closed. Afterwards, the data contained in the package is analyzed by the *specificCB()* function, as pictured in Figure 4.17. Once the analysis is completed, a message is sent to the SAN.

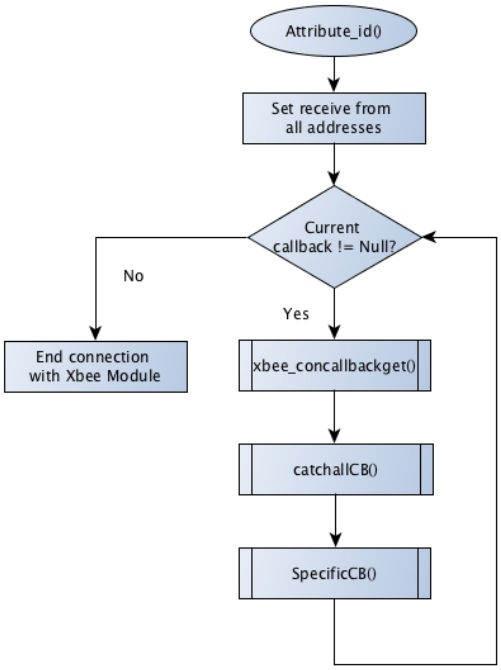


Figure 4.16: Flowchart of Xbee Communication Establishment

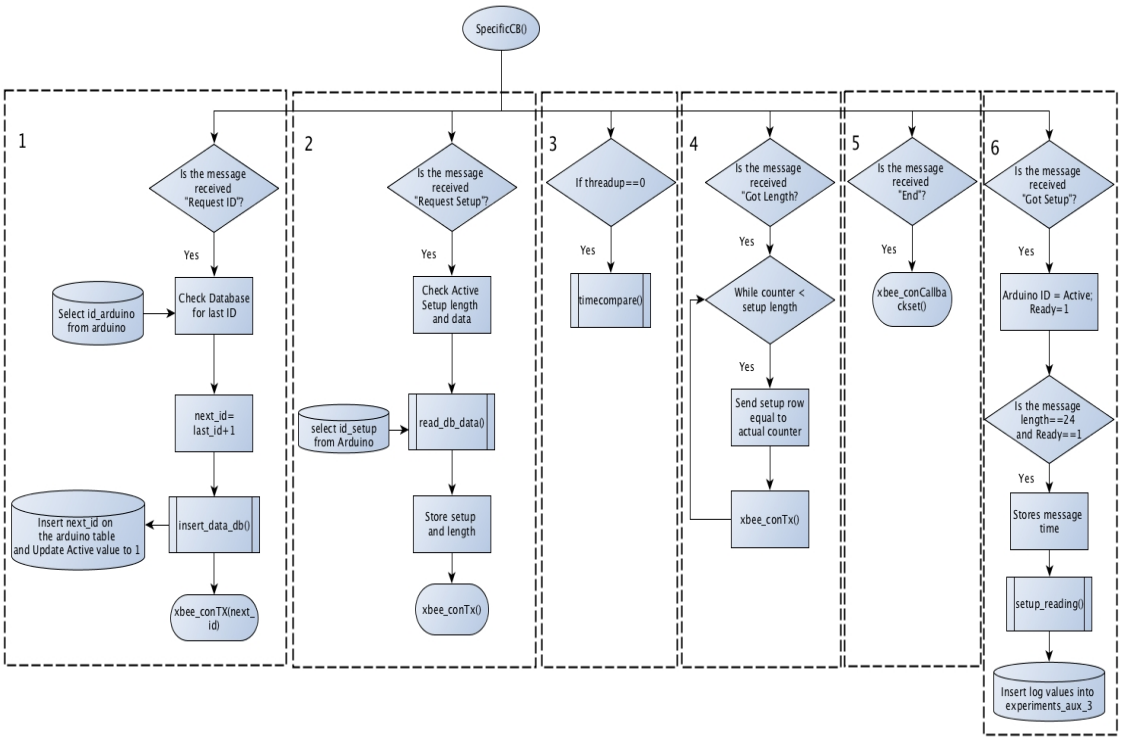


Figure 4.17: Flowchart of Message Reception Processing

The *SpecificCB()* is used to analyse the message received from a specific SAN, and according to the message received, there are multiple paths which to follow, pointed out with numbers in Figure 4.17.

For each path, the following actions take place:

1. When a message requesting an ID, the *read_db_data()* is called, which queries the Arduino Table on the PostgreSQL RDBMS, where it retrieves the Last ID value assigned to a SAN. That value will then be incremented by one, and stored on the Arduino Table, by using *insert_data_db()*;
2. When a message requesting a Setup is received, *read_db_data()* is called, and a query is made on the Arduino Table, checking if the SAN is currently active, and if a Setup was assigned to it. If so, the Setup ID and its length is retrieved and are sent to the SAN;
3. *timecompare()* is called, where the actual time of CCN is compared to the last message reception time from a given SAN ID. Once that timer goes over an established time, which was defined as 30 seconds, that SAN active flag is set to zero and the *insert_data_db()* is then called, updating the state value of that SAN ID in the database to 0 meaning that communications to it were lost while it was still running an experiment.
4. After the SAN sends the message "Got Length", confirming that it has correctly received the Setup ID and its length, a query is made to the RDBMS, selecting all the rows concerning the chosen Setup ID. Afterwards, all the rows are sent to the SAN;
5. Once a message "End" is received, the callback established to that specific SAN is ended;

6. After message "Got Setup" is received, the *insert_data_db()* function is called, and the row on Arduino table containg that SAN's ID (id_arduino) is updated, with the Active value changing to 1. Afterwards, it is expected that the next message from the SAN is a log data packet 4.2, and so the message length is checked, and if confirmed, the message time is stored, and the *setup_reading()* function, responsible for analysing the log data packet and extracting its information is called. After the information is extracted, it is stored on the experiments_aux_3 table, by calling the *insert_data_db()* function.

The working principle of the *insert_data_db()* function consists on:

- Receiving a char array with the database query;
- Inserting in the database, by use of *PQexec()* function;

Meanwhile, every time information needs to be retrieved from the database, the function *read_db_data()* is called. Its implementation is depicted on Figure 4.18.

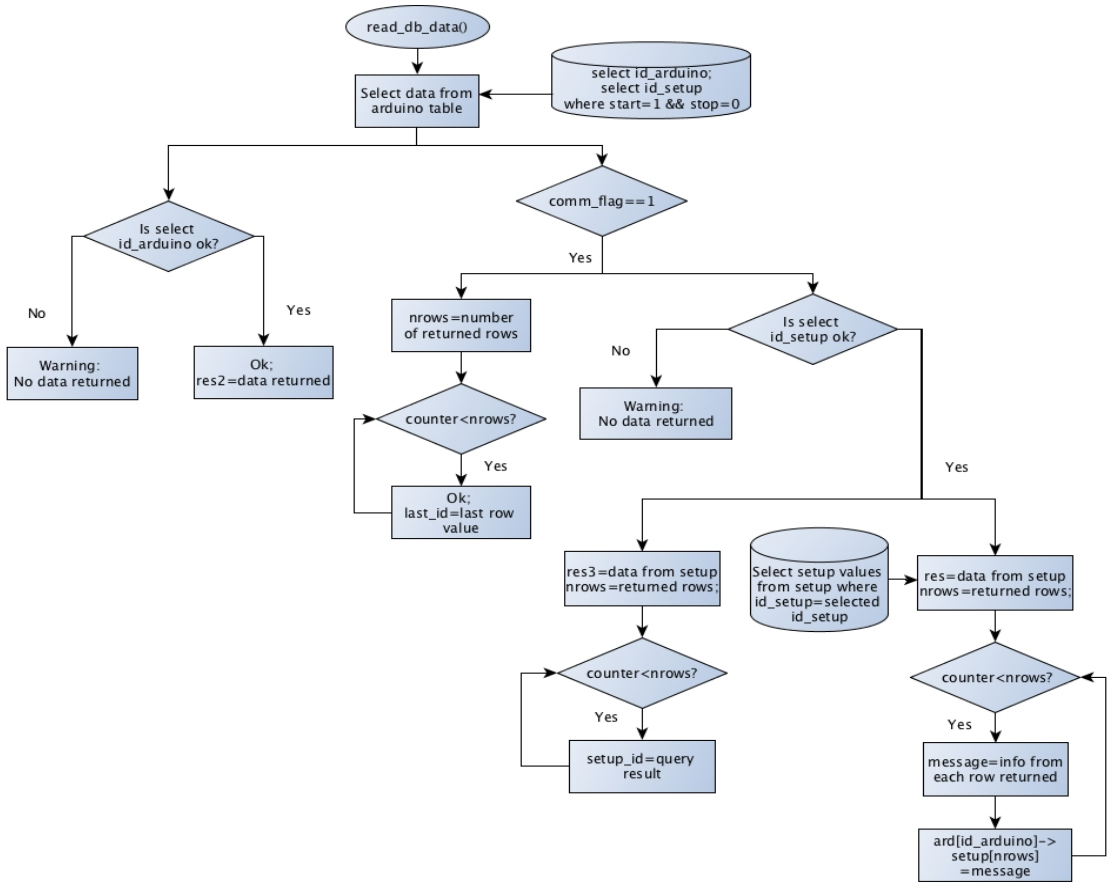


Figure 4.18: Flowchart of database queries

4.5 Sensor and Actuator Node

This section focuses on the implementation of the Sensor and Actuator Node, which is responsible for monitoring and controlling an experiment. Along this chapter, details on its behaviour, and firmware developed are provided.

4.5.1 Arduino Uno/Mega

The SAN was previously implemented in an *Arduino Mega*, and was able to run experiments by following a setup stored on its SD Card. This setup consisted on specific actions, such as turning on/off a water pump or an infrared light, on a specific time, to achieve a specific temperature or water level target. The main objective consisted on building upon this, enabling the user to send the experiment setups to the SAN via the System Interface, which removed the constraint of having to physically access the experiment site in order to change the setup of any given SAN. An example of a setup is presented in Table 4.1

Table 4.1: Setup Example

	Setup					
Row	Setup ID	SAN ID	Date RTC	Temperature	Tide	Lights
1	1	Null	1466305200	20	0	1
2	1	Null	1466305320	20	0	0
3	1	Null	1466305440	20	0	1
4	1	Null	1466305560	20	0	0
5	1	Null	1466305680	20	0	1
6	1	Null	1466305800	20	0	0
7	1	Null	1466305920	20	0	1

4.5.2 Xbee Library

As with the BBB, the implementation of the API Mode makes use of a Library [30], officially backed by Digi, which provides users with a simpler way of interacting with the Xbee Module for the Arduino. However, this created a constraint on the system, as this library only works if the Xbee is configured in API Mode 2 with escaped characters. Even though more functions available in the <*xbee.h*> are used in the development of the firmware, these are the essential ones for the use of the Xbee module:

- *xbee.readPacket()* - Reads data, if it is available. A timeout in miliseconds can be specified, after which the function returns that there is no data available;
- *rx.getDataLength()* - Returns the length of the message contained in the packet;
- *rx.getData()* - Retrieves the message from the packet;
- *XBeeAddress64()* - Sets the address of the destination module;

- *addr64.setMsb()* - Sets the most significant bytes of the address;
- *addr64.setLsb()* - Sets the less significant bytes of the address
- *Tx64Request(addr64, data, sizeof(data))* - Sets the message to be sent, according the the address defined, that that that is to be included, and its size.
- *xbee.send()* - Sends a data packet defined in the *Tx64Request*;

4.5.3 Firmware

The main aim of the firmware developed for the SAN was for it to be able to communicate directly with the CCN, without foreign intervention. As such, each Xbee Module placed on the SAN can only send messages to the Coordinator, as its address is the only one defined in the code.

The required features were implemented through a state machine and the states' progression is displayed in Figure 4.19.

To fully understand the state machine, it is necessary to analyse each state individually:

1. During the first state (state=0) the SAN sends "Request ID" messages, in a loop, to the CCN, until it receives an answer with an ID, which triggers the next state. This can be seen on Figure 4.20;
2. On the second state (state=1), the SAN sends Setup requests, in a loop to the SAN, once it gets the length and a complete setup, as can be seen on Figure 4.21;
3. When the third state (state=2) is triggered, data from each profile array is stored in auxiliary variables;
4. On state four (state=3), the running of setup is started;
5. When state five (state=4) is triggered, a message *"Logging"* is sent to the CCN, informing it that it is running the setup;
6. On state six (state=5), it is checked if the next profile setup is due or not, and data is acquired
7. During state seven (state=6), the last data acquired is sent to the CCN. If the setup reached its end, a message saying *"End"* is sent to the CCN.

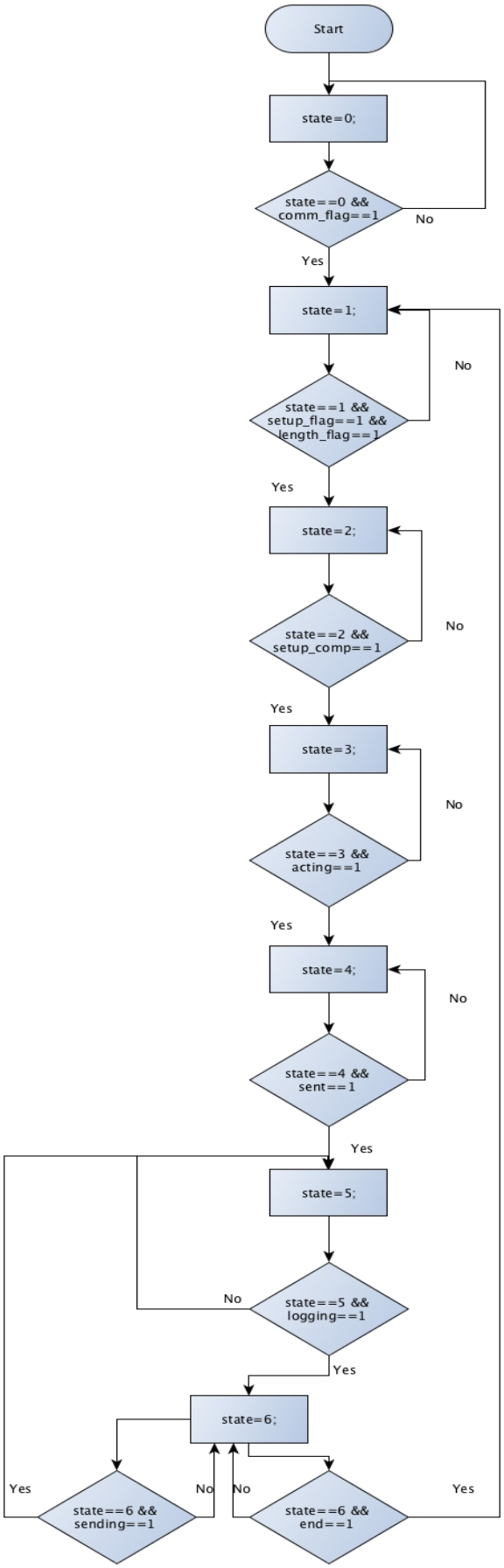


Figure 4.19: Flowchart of SAN State Machine

The *request_id()* function is used to send the "Request ID" message to the CCN, by calling *send_msg()* function. Once the message is sent, the *data_receive()* function is called, to check for incoming messages from the CCN. When a message is received, its first position (msg_aux[0]) is stored on an auxiliary variable. Then the length is checked, and if it is superior to zero, it is then stored, and a message saying "Got ID!" is sent back to the CCN, acknowledging the reception of an ID. Moreover, the communications up flag (comm_id) value is updated to one, enabling the transition from state=0 to state=1.

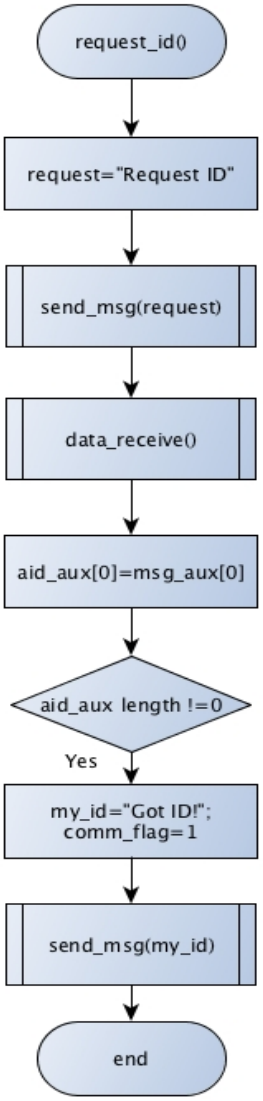


Figure 4.20: Flowchart of SAN ID Request

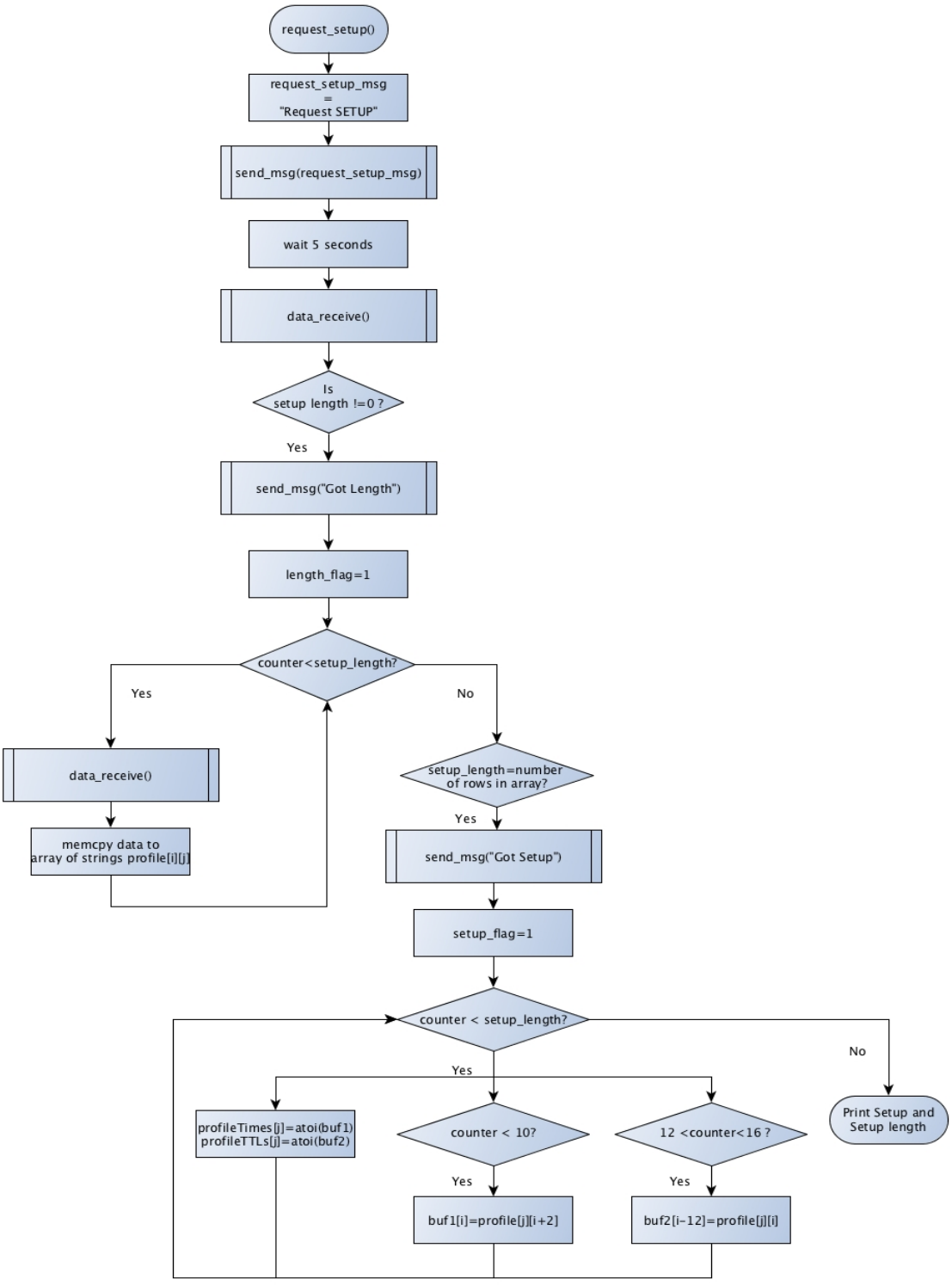


Figure 4.21: Flowchart of SAN Setup Request

The initial behaviour of the *request_setup()* function is similar to the *request_id()* function. A message saying "Request Setup" is sent to the CCN using *send_msg()*, a delay of five seconds is set and then it will check if the setup length was received. If so, the length is stored in a variable

called setup_length, and a message saying "Got Length" is sent to the CCN. Afterwards, a flag which states that the length was received (length_flag) is updated to one.

Then, as long as the counter is inferior to setup_length, the *data_receive()* is called, and each time a string containing one of the setup rows is received, it is stored in an array of strings (profile[i][j]), with the i stating the row, and j the position of each value to be stored in that string.

When the reception of the setup is complete, the values concerning the time since 1970 are stored in buf1, and the values concerning the Temperature, Tide and Light are stored in buf2. Then they are copied to profileTimes and profileTTLs, which are variables that were part of the original firmware developed by CIBIO, and essential to the behaviour of the firmware.

All these communications to the CCN require two functions:

1. *send_msg()* which expects a String to be passed on as an argument, however, given that *xbee.send()* requires messages to be a *uint_8* variable type, a conversion from *String* to *uint_8* has to be made. The algorithm is as presented in Figure 4.22.
2. textitdata_receive() on which the *xbee.readPacket()* is called, and a timeout of 5 seconds is defined, during which it waits for communications from the CCN. It was implemented as presented in Figure 4.23.

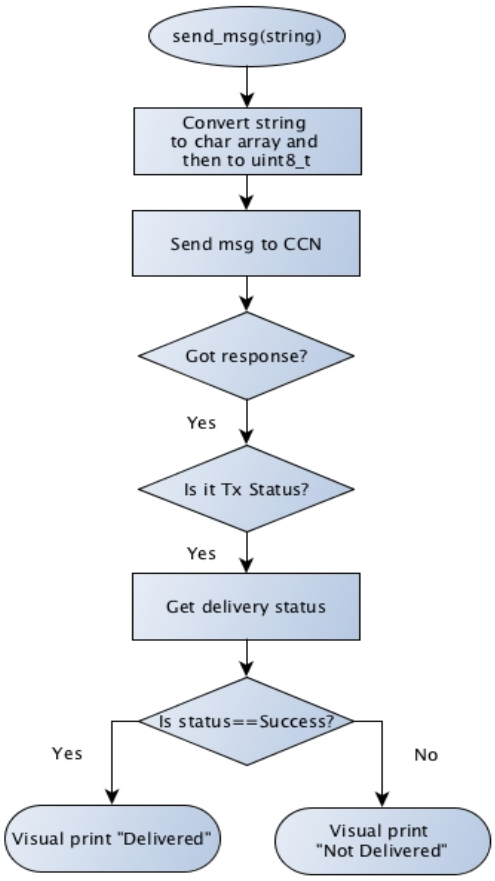


Figure 4.22: Flowchart of SAN Message Sending

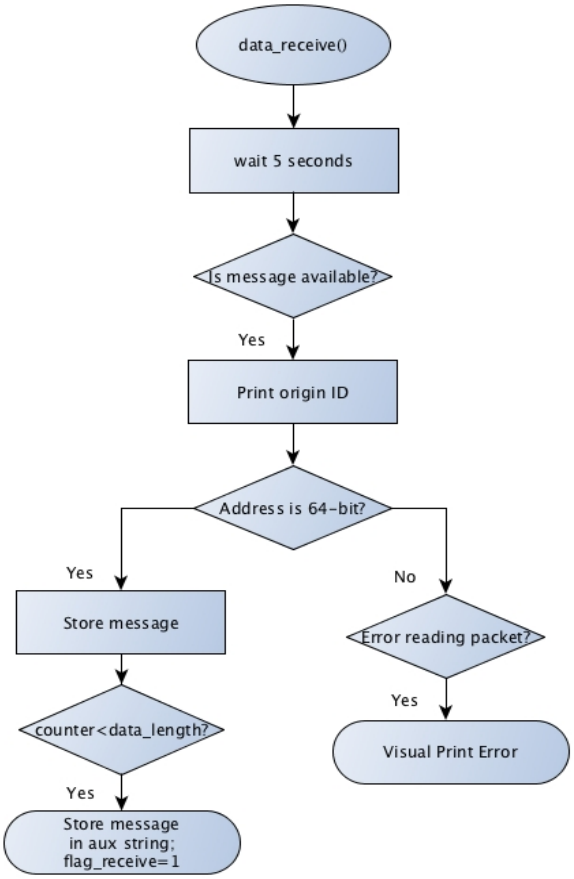


Figure 4.23: Flowchart of SAN Message Receive

4.6 System Interface

The conception of the System Interface consisted on the development of a Website, on which the user controls and monitors the experiments. To achieve this, an architecture was developed, which is thoroughly in this section. Moreover, details on the implementation of the architeture are also detailed.

4.6.1 Website Architecture

Taking into consideration the requirements presented on Chapter 3.4, use cases where designed, in order to represent the actions that an user is allowed to take, after accessing the web page.

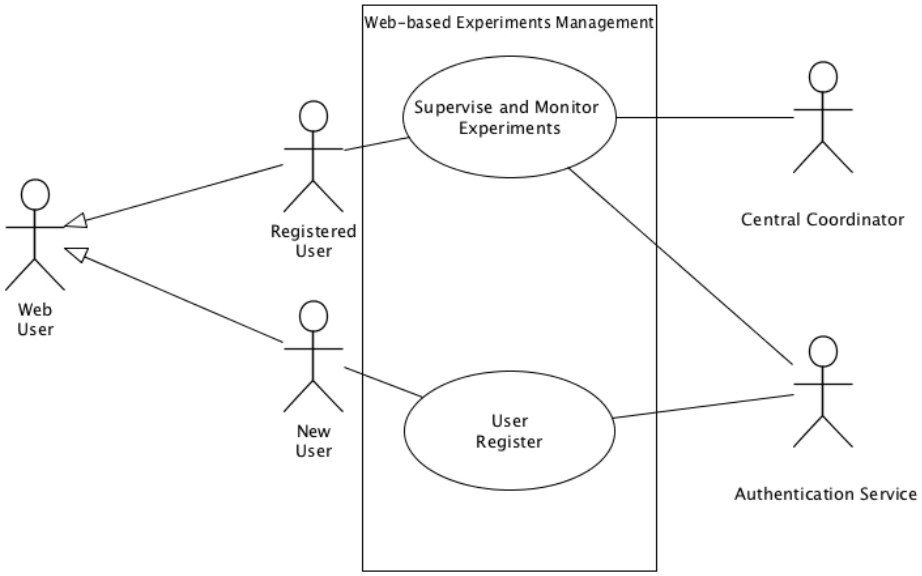


Figure 4.24: Web-Page Use Case Diagram

As shown on Figure 4.24, once the web page is accessed, the user can either register, or log in. In both, the data is passed via POST methods, so that the information is not forwarded via the web page link, as the passwords must be protected. In order to increase the system security, the passwords are stored with Secure Hash Algorithm 1 (SHA-1) encryption algorithm, so that even if someone accesses the database directly, they will not be able to immediately know a user's password. When the user logs in, the web page automatically encrypts the password inserted, and compares it to the encrypted password stored on the database.

Once the user logs in, there are two main actions he can take, either Supervise and Monitor the Experiments, or view and analyse the data acquired and stored. Such actions are presented on Figure 4.25 and Figure 4.26.

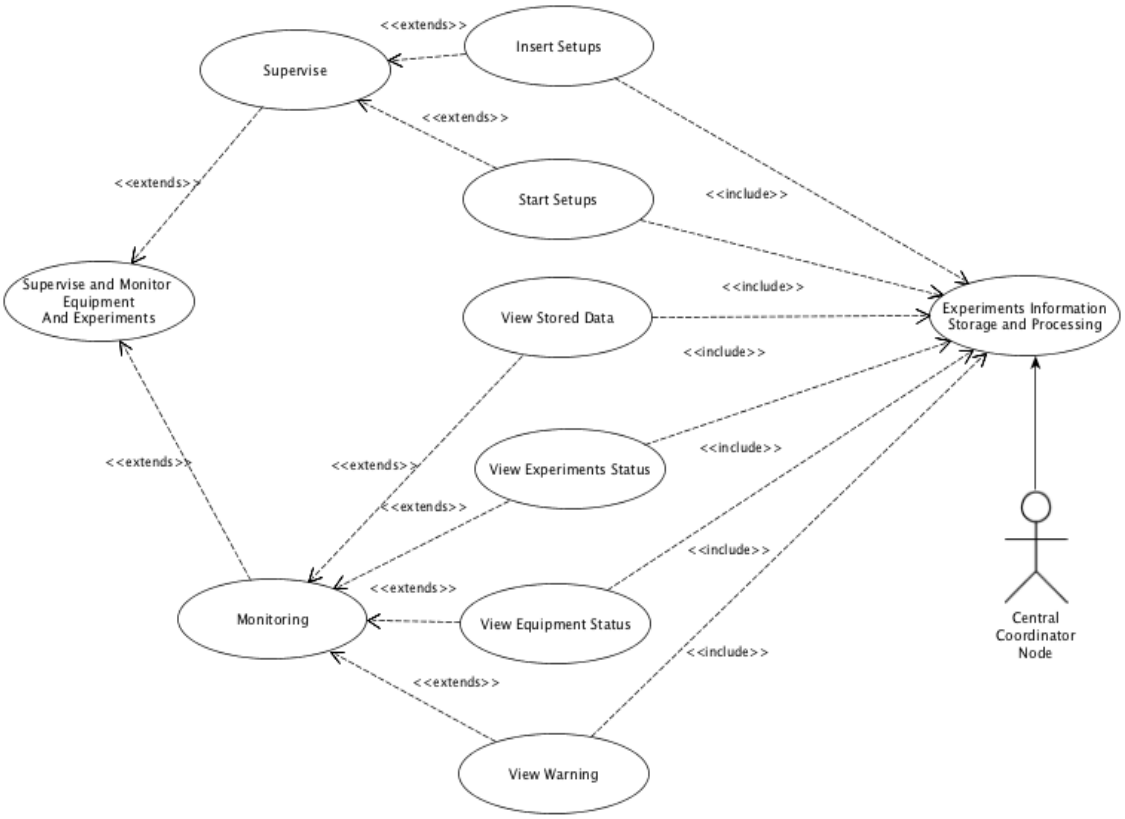


Figure 4.25: Experiments and Supervise Use Case Diagram

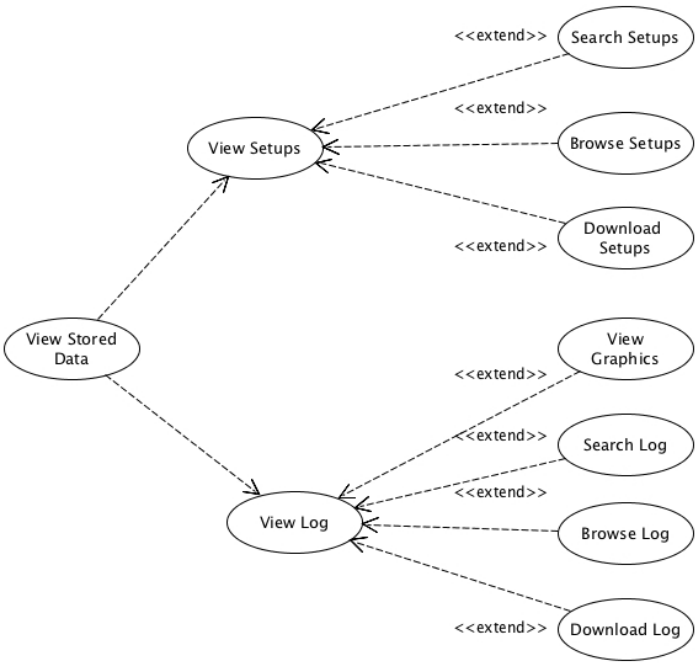


Figure 4.26: View Stored Data Use Case Diagram

The final web page architecture, in which all the use cases are available, is presented in Figure 4.27.

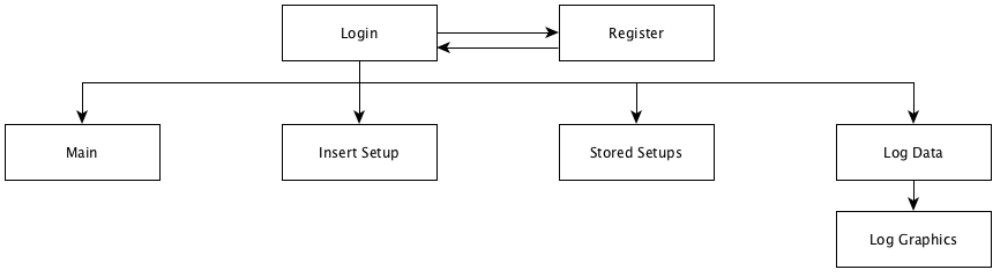


Figure 4.27: Web-site Architecture

To guarantee that no one can access the web pages without having logged in, after the user enters the main page, a new session is stared, by using *session_start()* PHP function. If a session has not been started, the user will be redirected to the log in page.

For further comprehension of the architecture shown on Figure 4.27, each page in specific must be analysed.

The main web page displays the following:

- Active SANs;
- Active Setups;
- Last Message Received, which is refreshed each time a new message is stored on the RDBMS;
- A selector used for starting new experiments, with the option of choosing a specific SAN and a specific Setup, amongst those active;
- System Alerts, at this time consisting of the communications' state;

On of the most important issues on the main page is to avoid the sending of incomplete Setups to the SANs. To solve this issue, a special query to the database was implemented, as shown on Listing 4.3:

```
1 "select distinct setup.id_setup, setup_aux.length_setup from
    setup_aux right join setup on setup.id_setup=setup_aux.id_setup
    where (select distinct count(*) from setup where setup.id_setup=
    setup_aux.id_setup)=setup_aux.length_setup order by id_setup"
```

Listing 4.2: SQL Query For Displaying Setup Available

Basically, it consists on comparing the length of the setup, stored in the auxiliary table, with the number of rows stored in the setup table with that specific ID. Finally, Alerts are displayed, such as failure in communications, temperature offsets and water levels different from the expected.

Carrying on to the Setup Insert page, both Setup ID and Setup Length are inserted in the database, on the setup auxiliary table, and the setup data inserted on the setup table.

On the Stored Setups page, the information stored on the *setup_aux* table is presented, so that the user can check if the data is stored correctly.

Finally, there is the Log Data page, were the information stored on the *experiments_test_3* table is displayed. Furthermore, there is the Log Graphics page, were the data stored on the *experiments_test_3* table is displayed in Graphics, shown the variation in temperature, the Lights Actuator level, and the Water Level, according to the timestamp.

4.6.2 RDBMS Interaction

In order to fulfill the Experiments Information Storage and Processing case, an establish a connection to the PostgreSQL RDBMS is needed. To accomplish so, a *.php* file was built, containing a class, shown in Listing 4.2, that deals with the communications and where all the functions needed to query the RDBMS were implemented. Consequently, each new sub page must call a new occurrence of said class.

```
1 typedef class teste3{
2   private $host = '192.168.1.164'; //postgresql host address
3   private $porta = 5432; //Port in which it is conneccted
4   private $dbnome = 'postgres'; //Database name
5   private $user = 'postgres'; //Database username
6   private $pass = 'jonaspombo'; //Database Password
7   public $liga; // Variable which allows the access to the private variables, by
        the functions
8   }
```

Listing 4.3: PostgreSQL Connection Class

Each query made to the RDBMS is done through a prepared statement, which is then executed. As such, the web page containing the class described on Listing 4.2 also has the functions used to send queries to the database, which are:

- *login($username, $password)*: Receives the username and password, and compares the values received with those stored on the database;
- *registar_utilizador($username, $email, $password)*: Inserts the username, email and password on the Login Table, given that no parameter is repeated;
- *logout()*: Ends the session;
- *active_setup()*: Returns the SAN ID and the Setup ID that it is running;
- *active_arduino()*: Checks the SANs which are running an experiment;
- *available_arduino()*: Checks the SANs which are currently connected with the CCN and without a Setup ID associated;
- *begin_experiment($setupid, $arduinoid, $start)*: Updates the start value associated with a specific SAN ID, allowing it to start the experiment;
- *last_message()*: Selects the row stored on the *experiments_test_3* table;

- *start_setup()*: Checks the setups on which the specified length is equal to the number of rows stored for it;
- *graphic_temp()*: Selects data from the *experiments_test_3* relating to the chosen SAN and Setup ID, in order to draw a plot;

The following step consisted on building a web page dedicated to formulary processing, thus making it possible to know which action the user performed, eg. pressing a specific button, which then triggers a function associated with the action defined on the button.

4.6.3 Data Visualisation

4.6.3.1 Representing Data in Tables

To represent data in tables, *DataTables* library [31] was used. This library makes use of *JavaScript* to query the database for the logging data present on Experiments Table. This data is shown on a single table, with the user being able to search specific data and also to download the data present on the tables in various formats, namely *xml*, *csv*, and *pdf*.

The *JavaScript* code implemented for the retrieving of the data to be presented is shown on the Listing 4.4.

```
$('#example').DataTable( {
  dom: 'Bfrtip',
  lengthMenu: [
          [ 10, 25, 50, -1 ],
          [ '10 rows', '25 rows', '50 rows', 'Show all' ]
      ],
      buttons: [
          'copy', 'csv', 'excel', 'pdf', 'print','pageLength'
      ],
  'processing': true,
  'paging': true,
  'pagingType': 'full_numbers', //full
  'serverSide': true,
  'ajax': 'ejemplo_fact_process.php',
  columnDefs: [
    { width: 50, targets: [0, 1] },
    { width: 50, targets: [-1, -2]}
  ]
  } );
```

Listing 4.4: Javascript for Representing Data in a Table

To acquire the data for the tables, the *JavaScript* calls the file *ejemplo_fact_process.php* which sends a query to the database throught a JavaScript Object Notation (JSON), which is sent to another *php* file, which will then process the JSON and send the query to the database.

4.6.3.2 Real-Time Graphics

One of the system requirements was to be able to draw graphics of Temperature, Water Level and Light Actuation, according to the timestamps. This was done using the *pChart* class, which enables the creation of dynamic graphics in PHP [22, 32]. With this tool, it is possible to create various types of graphics, such as line or circular graphics.

In this specific case, the graphics are drawn according to the user selection of SAN and Setup IDs. Once the selection is made, *pChart* is called, in a separate web page, unknown to the user, which queries the database according to the selection of SAN and Setup IDs, saves the images on the server, which are then displayed on the graphics visualisation page.

4.6.3.3 Warnings Notification

In order to send warning to the users concerning erroneous system behaviour, the System Interface checks the parameters that are off and then sends an email to all the users. This was achieved with the help of *PHPMailer*, which is a "A full-featured email creation and transfer class for PHP" [33]. With this library it is possible to send emails without having the necessity of configuring a Simple Mail Transfer Protocol (SMTP), by making an authentication to an external email service, in this case a *Google Mail Account*, and using said service to send the email.

Chapter 5

Testing and System Validation

In this chapter, the validation of the system's requirements implemented will be presented. The test bed used in this chapter was comprised of a BeagleBone Black acting as a CCN, three Arduino Uno acting as SAN, a PC, running Windows 7, with Xampp serving as the Apache Server, where the PostgreSQL RDBMS and System Interface are hosted.

5.1 Communications between the CCN and SANs

To validate the communications between the CCN and the SANs, several tests are needed. First, there is the need to confirm that the behaviour between the CCN and a singular SAN is as expected. Subsequentially, it must be certified that the same behaviour applies when the CCN receives messages from several SANs. To do so, tests on the attribution of ID for one or more SANs were conduced, as well as the deployment of a single setup to a single SAN, and multiple setups, to multiple SANs. Lastly, the acknowledgment of lost communications, by the CCN side, needs to be ratified.

5.1.1 ID Attribution

This test consists on checking if, when a Request ID message is sent by a SAN, it acknowledges that the message was delivered, and cross-check the reception on the CCNs part. Assuming that the test occurs as expected, the CCN should reply to the SAN with an ID value.

5.1.1.1 Single SAN Connected

This specific test consisted on turning on one SAN, checking that the message was delivered to the CCN, and that a reply with an ID was sent.

Figure 5.1: Sending And Confirmation Of "Request ID" By The SAN

As seen on Figure 5.1, the message "Request ID" is displayed, as well as the acknowledgment that the message was successfully sent to the CCN, with the message "MSG ENTREGUE". However, this messages only tells us that the Xbee Module placed on the CCN got the message, and not that the CCN actually received and processed it.

To get the confirmation of the message's reception on the CCN's part, the terminal running on the CCN needs to be checked. Figure 5.2 shows the escaped read of the incoming message, and finally displays the sender's address, and the message received, after being processed, which is displayed next to "rx_P" and "rx_I". This confirms that the message "Request ID" was sucessfully sent from the SAN and received by the CCN.

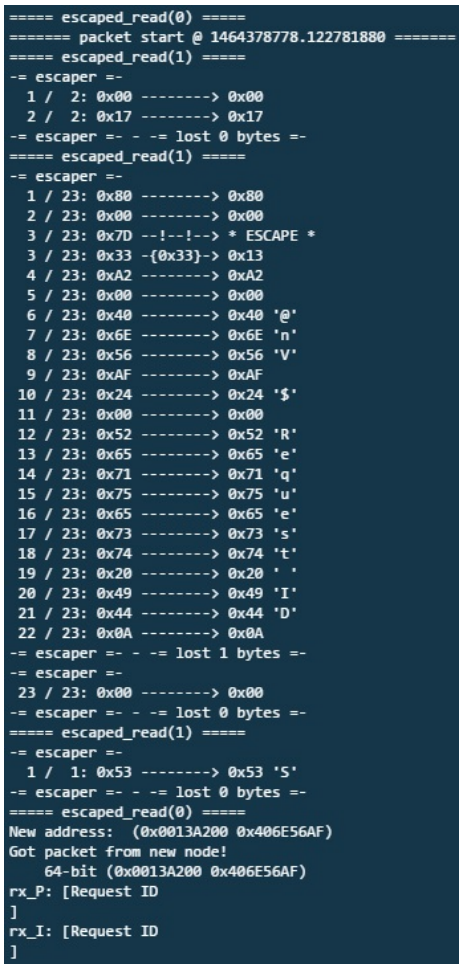


Figure 5.2: CCN Receiving "Request ID" Message

Afterwards, the CCN must check the Arduino Table on the PostgreSQL RDBMS, and assign the next ID available. Given that there was not any SAN previously connected to the CCN, the ID that sent to the SAN should be 1. As it can be seen on Figure 5.3, the ID is selected as expected, one, and the Arduino Table is successfully updated. Furthermore, it is confirmed that the message was successfully delivered to the SAN. This is done by looking at the tx value, which is equal to 0. That value is the expected return from the *xbee_connTx* function.

```
Tou a ler data da DB para atribuir ID
comm: 1
Insert_setup: select id_setup from arduino where id_arduino=0 and start=1 and stop=0
Ok
Flag read: 1
Debug
number of rows returned = 0
Arduino ID next ID: 1
ID Atribuido com sucesso
Aux: INSERT INTO arduino (id_arduino) VALUES ('1')

insert: 1
======= packet start @ 1464378778.273026964 =======
===== escaped_read(1) =====
-= escaper =-
  1 /  2: 0x00 --------> 0x00
  2 /  2: 0x03 --------> 0x03
-= escaper =- - -= lost 0 bytes =-
===== escaped_read(1) =====
-= escaper =-
  1 /  3: 0x89 --------> 0x89
  2 /  3: 0x01 --------> 0x01
  3 /  3: 0x00 --------> 0x00
-= escaper =- - -= lost 0 bytes =-
===== escaped_read(1) =====
-= escaper =-
  1 /  1: 0x75 --------> 0x75 'u'
-= escaper =- - -= lost 0 bytes =-
tx: 0
Aux: UPDATE arduino SET active='1', id_setup='0', start='0', stop='0' WHERE id_arduino='1'
```

Figure 5.3: CCN Attributing ID

On Figure 5.4, it can be seen that the Xbee is receiving a Tx message from a 64-bit address, since the API ID is equal to 128, and that the message contains the ID, which is one, as expected. Additionally, the SAN sends the acknowledgement message "Got ID!" to the CCN, and displays the ID assigned "My ID is: 1".

```
API ID: 128
A receber: 1

Data pck: Got ID!
Size data:8
~)3)36@nU_Got ID!'MSG ENTREGUE
My ID is: 1
My ID sent is: Got ID!
State:1
```

Figure 5.4: SAN Confirming The Reception Of ID Value

The recepetion of the acknowldegment message by the CCN is shown on 5.5. Bar the Address value, the analysis of this figure is in all similar to Figure 5.2.

```
======= packet start @ 1464378778.385082005 =======
===== escaped_read(1) =====
-= escaper =-
  1 /  2: 0x00 --------> 0x00
  2 /  2: 0x7D --!--!--> * ESCAPE *
-= escaper =- - -= lost 1 bytes =-
-= escaper =-
  2 /  2: 0x33 -{0x33}-> 0x13
-= escaper =- - -= lost 0 bytes =-
===== escaped_read(1) =====
-= escaper =-
  1 / 19: 0x80 --------> 0x80
  2 / 19: 0x00 --------> 0x00
  3 / 19: 0x7D --!--!--> * ESCAPE *
  3 / 19: 0x33 -{0x33}-> 0x13
  4 / 19: 0xA2 --------> 0xA2
  5 / 19: 0x00 --------> 0x00
  6 / 19: 0x40 --------> 0x40 '@'
  7 / 19: 0x6E --------> 0x6E 'n'
  8 / 19: 0x56 --------> 0x56 'V'
  9 / 19: 0xAF --------> 0xAF
 10 / 19: 0x24 --------> 0x24 '$'
 11 / 19: 0x00 --------> 0x00
 12 / 19: 0x47 --------> 0x47 'G'
 13 / 19: 0x6F --------> 0x6F 'o'
 14 / 19: 0x74 --------> 0x74 't'
 15 / 19: 0x20 --------> 0x20 ' '
 16 / 19: 0x49 --------> 0x49 'I'
 17 / 19: 0x44 --------> 0x44 'D'
 18 / 19: 0x21 --------> 0x21 '!'
-= escaper =- - -= lost 1 bytes =-
-= escaper =-
 19 / 19: 0x00 --------> 0x00
-= escaper =- - -= lost 0 bytes =-
===== escaped_read(1) =====
-= escaper =-
  1 /  1: 0xFB --------> 0xFB
-= escaper =- - -= lost 0 bytes =-
===== escaped_read(0) =====
Size: 1rx_P: [Got ID!]
rx_I: [Got ID!]
Is it got it? A: 10
```

Figure 5.5: CCN Receives "Got ID" Message

5.1.1.2 Two SANs Connected

This test consisted on activating a second SAN, and check if the ID was assigned as number two.

5.1.2 Experiment Deployment

In order to test the running of an Experiment, two distinct setups were developed. Taking into consideration that this is a theoretical experiment, it does not use the water tank and actuators on which the system is to be integrated into. Instead, it makes use of a LM35 Precision Centigrade Temperature Sensor as the Temperature Sensor, and a Light Emitting Diode (LED) representing the Lights. As there was no physical way to actually variate the temperature, the temperature setpoint will be set as 20, which was randomly chosen. Moreover, no component was used to represent the tide, and as such, its value will be left to zero.

The First Setup, shown on Table 5.1, has the ID one, and it was set to start at 4:00 AM, GMT+1 Time, with each step of the setup being taken every two minutes. The times are displayed in seconds since 1970, which is the expected time form. The Second Setup, which is shown on Table 5.2 follows the same logic, although the start time was set to 5:00 AM, GMT+1 Time, since

the Real-Time Clock (RTC) placed on that specific SAN is running an hour late. Seeing that it is only being tested if the setup is acquired by the SAN and the LED is being correctly turned on and off, besides of taking temperature readings, there was no actual need to calibrate the RTC's Time.

Table 5.1: Setup ID Number 1

	Setup					
Row	Setup ID	SAN ID	Date RTC	Temperature	Tide	Lights
1	1	Null	1466305200	20	0	1
2	1	Null	1466305320	20	0	0
3	1	Null	1466305440	20	0	1
4	1	Null	1466305560	20	0	0
5	1	Null	1466305680	20	0	1
6	1	Null	1466305800	20	0	0
7	1	Null	1466305920	20	0	1

Table 5.2: Setup ID Number 2

	Setup					
Row	Setup ID	SAN ID	Date RTC	Temperature	Tide	Lights
1	2	Null	1466308800	20	0	0
2	2	Null	1466308920	20	0	1
3	2	Null	1466309040	20	0	0
4	2	Null	1466309160	20	0	1
5	2	Null	1466309280	20	0	0
6	2	Null	1466309400	20	0	1
7	2	Null	1466309520	20	0	0

Since it was hard to capture the data on the terminal for both the SANs, only the communications between the SAN with ID 2 and the CCN are going to be presented. Nonetheless, the behaviour was similar for both SANs.

On Figure 5.6, it can be seen that the SAN sends a "Request SETUP" message to the CCN, and once more, confirms that the CCN's Xbee Module received the message. Figure 5.7 shows the CCN's "Request Setup" message reception. Furthermore, on Figure 5.6 it is proved that a Setup was assigned, in this case, Setup 2 was the choice, and its length was correctly received.

```
State:1
Tou a pedir SETUP
Data pck: Request SETUP
Size data:14
~     }3◦ @nU_ Request SETUP MMSG ENTREGUE
Tou a espera de receber SETUP
API ID: 128
A receber: 7

Size of setup:7
Data pck: Got Length
Size data:11
~     }3◦ @nU_ Got Length ;MSG ENTREGUE
```

Figure 5.6: SAN Number Two Requesting And Acknowledging the Reception Of A Setup Length

```
======= packet start @ 1464378780.999034756 =======
===== escaped_read(1) =====
-= escaper =-
  1 /  2: 0x00 --------> 0x00
  2 /  2: 0x19 --------> 0x19
-= escaper =- - -= lost 0 bytes =-
===== escaped_read(1) =====
-= escaper =-
  1 / 25: 0x80 --------> 0x80
  2 / 25: 0x00 --------> 0x00
  3 / 25: 0x7D --!--!--> * ESCAPE *
  3 / 25: 0x33 -{0x33}-> 0x13
  4 / 25: 0xA2 --------> 0xA2
  5 / 25: 0x00 --------> 0x00
  6 / 25: 0x40 --------> 0x40 '@'
  7 / 25: 0x6E --------> 0x6E 'n'
  8 / 25: 0x56 --------> 0x56 'V'
  9 / 25: 0xAF --------> 0xAF
 10 / 25: 0x24 --------> 0x24 '$'
 11 / 25: 0x00 --------> 0x00
 12 / 25: 0x52 --------> 0x52 'R'
 13 / 25: 0x65 --------> 0x65 'e'
 14 / 25: 0x71 --------> 0x71 'q'
 15 / 25: 0x75 --------> 0x75 'u'
 16 / 25: 0x65 --------> 0x65 'e'
 17 / 25: 0x73 --------> 0x73 's'
 18 / 25: 0x74 --------> 0x74 't'
 19 / 25: 0x20 --------> 0x20 ' '
 20 / 25: 0x53 --------> 0x53 'S'
 21 / 25: 0x45 --------> 0x45 'E'
 22 / 25: 0x54 --------> 0x54 'T'
 23 / 25: 0x55 --------> 0x55 'U'
 24 / 25: 0x50 --------> 0x50 'P'
-= escaper =- - -= lost 1 bytes =-
-= escaper =-
 25 / 25: 0x00 --------> 0x00
-= escaper =- - -= lost 0 bytes =-
===== escaped_read(1) =====
-= escaper =-
  1 /  1: 0x59 --------> 0x59 'Y'
-= escaper =- - -= lost 0 bytes =-
rx_P: [Request SETUP]
rx_I: [Request SETUP]
```

Figure 5.7: Reception of "Request Setup" Message By The CCN

Then, on Figure 5.8, the complete setup reception by the SAN is shown, alongside with the sending of the acknowledge message of "Got Setup" once the full setup was received.

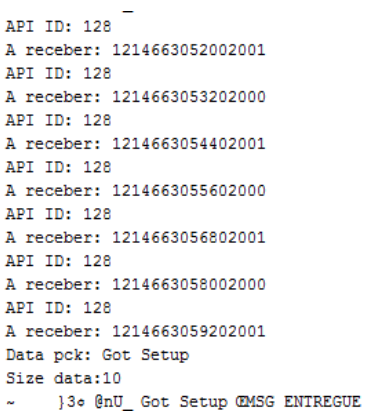


Figure 5.8: SAN Number 2 Receiving Setup And Acknowledge Message

It can be seen on 5.9 by comparing the Profile Times, and Profile TTLs with the values shown on Table 5.2 that the Setup was correctly received, with the values being stored correctly, as presented after "Profile:".

```
Profile Times: 1466305200
Profile TTLs: 2001
Profile Times: 1466305320
Profile TTLs: 2000
Profile Times: 1466305440
Profile TTLs: 2001
Profile Times: 1466305560
Profile TTLs: 2000
Profile Times: 1466305680
Profile TTLs: 2001
Profile Times: 1466305800
Profile TTLs: 2000
Profile Times: 1466305920
Profile TTLs: 2001
Setup length: 7
Profile:1214663052002001
1214663053202000
1214663054402001
1214663055602000
1214663056802001
1214663058002000
1214663059202001
```

Figure 5.9: SAN Number 2 Stored Setup

The sending and reception of Logging data is demonstrated on Figures 5.10 and 5.11. Due to the difficulty of acquiring the same message being sent and received, the messages shown on both pictures display similar images, but with different timestamps.

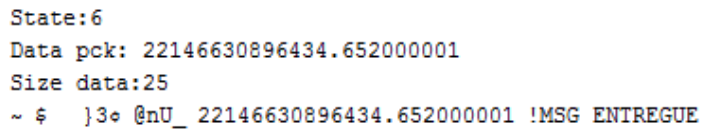


Figure 5.10: SAN Number 2 Sending A Log Message

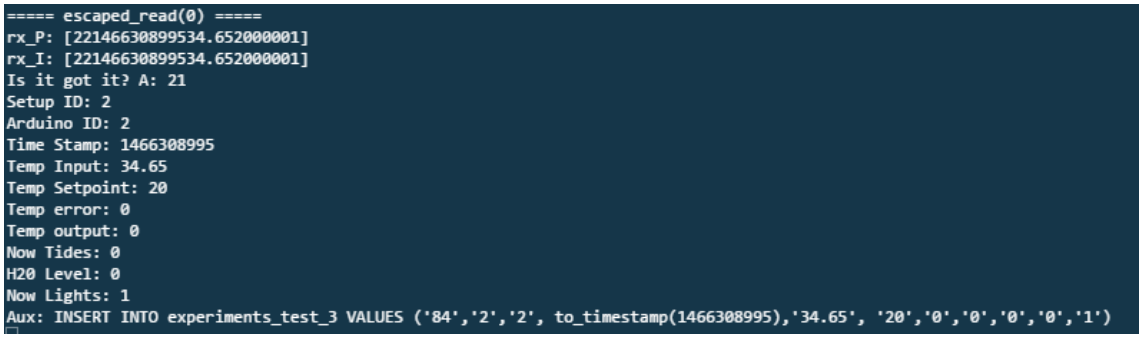


Figure 5.11: CCN Receiving a Log Message From SAN Number 2

5.1.3 Acknowledging Loss of Communications

One of the main points to be validated is the reaction of the CCN to the Loss of SANs Communications. This test procedure consisted of manually switching the switch present on the Wireless SD Shield, placed on each of the SANs, from Micro to USB, which will disable the UART communications via the Xbee Module, thus simulating the loss of communications. In this test, first the SAN number 1 communications were disabled, and after 30 seconds, the SAN number 2 communications were disabled.

On Figure 5.12 it can be seen that the SAN failed to deliver the message to the CCN, since the acknowledgment message "MSG Entregue" was not displayed.

Figure 5.12: SAN Number 1 Failing to Deliver A Log Message

The Loss of Communications shall be displayed on the System Interface, and such happens, as expected, as can be seen on Figures 5.13 and 5.14.

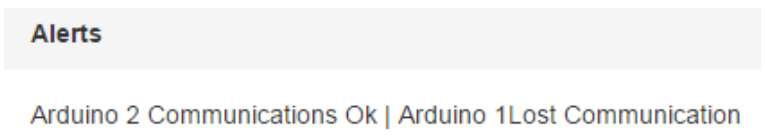


Figure 5.13: System Interface Displaying Lost Communications From SAN Number 1

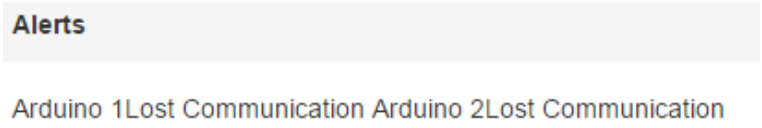


Figure 5.14: System Interface Displaying Lost Communications From All SANs

During the time in which the SANs communications are down, no information is displayed on the CCN's terminal. However, once the communications are re-established, the CCN, once it receives a message with a Log Data, it shall update that SAN ID Active Value back to one. Such behaviour is shown on Figures 5.15 and 5.16.

```
===== escaped_read(0) =====
rx_P: [11146630643334.652000000]
rx_I: [11146630643334.652000000]
Is it got it? A: 22
Setup ID: 1
Arduino ID: 1
Time Stamp: 1466306433
Temp Input: 34.65
Temp Setpoint: 20
Temp error: 0
Temp output: 0
Now Tides: 0
H20 Level: 0
Now Lights: 0
Aux: INSERT INTO experiments_test_3 VALUES ('403','1','1', to_timestamp(1466306433),'34.65', '20','0','0','0','0','0')
Aux: UPDATE arduino SET active='1' WHERE id_arduino='1'
```

Figure 5.15: Communications Between CCN And SAN Number 1 Restored

```
===== escaped_read(0) =====
rx_P: [22146631007934.652000000]
rx_I: [22146631007934.652000000]
Is it got it? A: 21
Setup ID: 2
Arduino ID: 2
Time Stamp: 1466310079
Temp Input: 34.65
Temp Setpoint: 20
Temp error: 0
Temp output: 0
Now Tides: 0
H20 Level: 0
Now Lights: 0
Aux: INSERT INTO experiments_test_3 VALUES ('420','2','2', to_timestamp(1466310079),'34.65', '20','0','0','0','0','0')
Aux: UPDATE arduino SET active='1' WHERE id_arduino='2'
```

Figure 5.16: Communications Between CCN And SAN Number 2 Restored

5.2 Communications between the CCN and DBMS/System Interface

The System Interface is supposed to have the ability to display the logging messages, draw plots from the data, show warnings for lost communications, as well as the state of the SANs and which setups they can run.

5.2.1 System State

To verify that the System Interface correctly displays the available Setups and Information Regarding the SANs state, two steps were taken. First, after storing the Setup Values, the System Interface was checked so as to see that it displayed the Setup ID and Length Correctly. Finally, after the SANs were running the Setups, it was checked if it was displaying the correct SAN ID (identified in the pictures as Arduino ID) with the correct Setup ID following. As it can be seen on Figures 5.17 and 5.18, the data is displayed correctly.

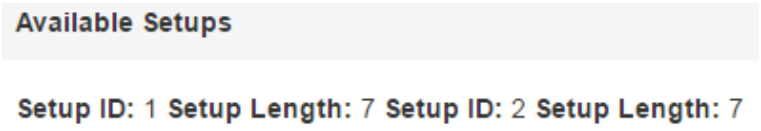


Figure 5.17: Available Setups

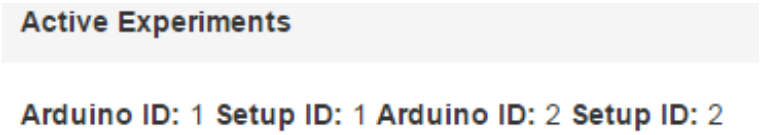


Figure 5.18: SAN Running Experiments

5.2.2 Data Storage

For the sake of certifying that the logging messages were being correctly stored and displayed by the System Interface, a check was made on the table which displays the log data, which can be seen on 5.19. This image corroborates that the data was being correctly sent, as it displays messages received from both SANs, within the expected timestamps from the setup.

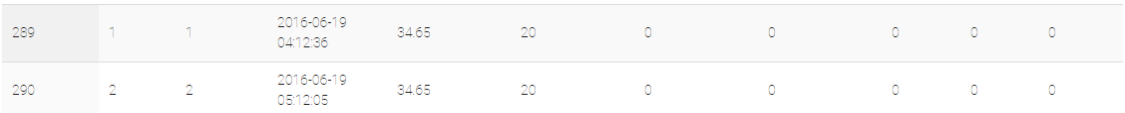


Figure 5.19: Stored Log Data

In addition, this is certified by the viewing of the Last Message Received Tab present on the Main Page of the System Interface, which is shown of Figure 5.20

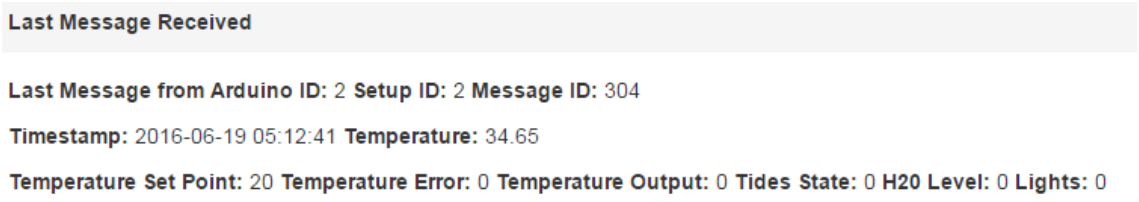


Figure 5.20: Displaying Last Message Received

5.2.3 Data Visualization

Since the System Interface shall be able to correctly represent data acquired in graphics, according to the timestamps, a test was developed in order to validate the mentioned requirement. To do so, the data related to the SAN number 1 was downloaded in a .xls file, and a plot was made using Excel, as is presented in Figure 5.22. On Figure 5.21, the graphic displayed by the System Interface is shown.

Figure 5.21: Plot of SAN ID 1 Running Setup ID 1 on the System Interface

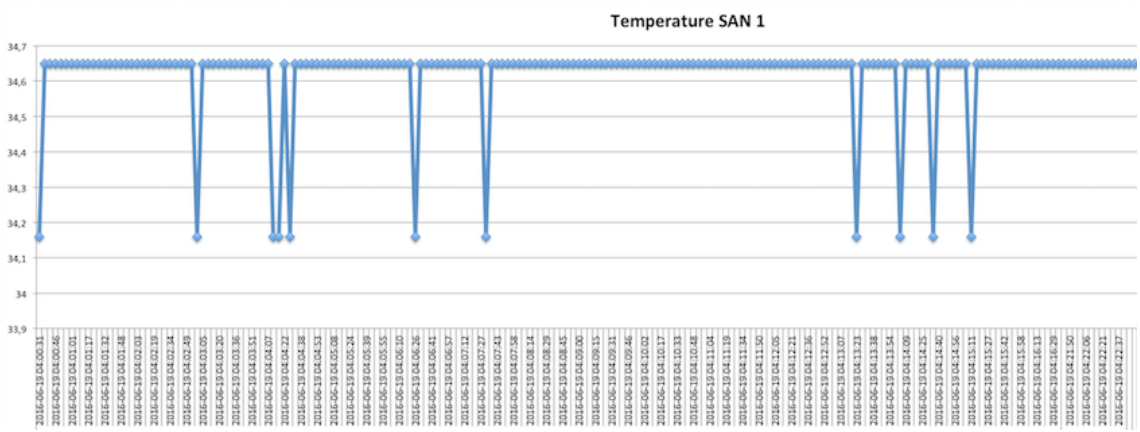


Figure 5.22: Plot of SAN ID 1 Running Setup ID 1 with Data Downloaded From Data Tables

As it can be concluded from the comparison between the two, even though both are fairly similar, there are some temperature variations present on Figure 5.22 that cannot be seen on Figure 5.21, meaning that the requirement is not completely fulfilled.

Chapter 6

Conclusions and Future Work

This dissertation focused on the development of a system capable of remote monitoring and control of ecological experiments for CIBIO. The main goal was to achieve a system that would allow the researchers to conduce experiments and control their evolution remotely, through the Internet.

The project started with the assessment of the requirements of the problem. Then, a study on various system's architectures that would allow for the implementation of the required system as made, such as the WSN, WSAN, NCS and Remote Labs. Afterwards, a comparative research was made in order to find the best solution for the architecture, as well as to find the best hardware and software for its development.

Finally, and taking into consideration the system requirement's presented in section 3.2 the solution proposed is an architecture with three levels. First, the bottom tier, which comprises several Sensor and Actuator Nodes, with the role of supervising experiments. Then, the middle tier, which has a Central Coordinator Node, responsible for establishing a bridge between the sensors and the System Interface, communicating with the bottom tier via a wireless communications protocol. In order to achieve this, the following choices were made:

- For the communications between the SAN and the CCN, protocol chosen was **IEEE 802.15.4** as it ensured the possibility of connecting 9 or more nodes, without adding extreme complexity to the network;
- A **BeagleBone Black** was chosen for the role of Central Coordinator Node, as its hardware and software - a Linux Distribution - is capable of handling the processing of information from several SANs and storing on the RDBMS;
- To store the information regarding the system, from setups, to log data, **PostgreSQL RDBMS** was employed;
- A **Web Page** was developed as the System Interface, using **PHP** and **HTML 4 and 5**, where users can access and manage the Experiments;
- An **Apache web server** was used in order to host the PostreSQL RDBMS and the Web Page on a Local Area Network.

Even though theoretical results are presented, the system's validation is still an ongoing process, since it is yet to be deployed on the Laboratories at CIBIO. However, it is possible to conclude

that the system is able to handle more than one SAN at a time, with no loss of information between the CCN and the DBMS, given optimal conditions, such as a reliable Internet connection, and absence of interference with the Xbee Modules.

From a development point of view, it proved to be quite hard to incorporate the previous hardware into the final system, taking into consideration that the software that it runs follows a certain structure, and makes use of determined variable types, making the development of code for the incorporation of the Xbee Modules much harder, since there was the need for converting data in order to match the expected variables by the functions which were previously running on the SAN.

Besides of the development of the system, one of the goals for this Dissertation was to write a paper in order to selected for OCEANS'16 MTS/IEEE Monterey Conference. At the present, the abstract submitted has passed the review for inclusion in the regular technical program.

6.1 Future Work

The main goals of this dissertation were to create a system that would allow the retrieval of data from the experiments, begin new experiments, and receive warnings on their state. As such, prototype capable of such was developed, however, a operational final system is yet to be completed, in order to validate the system's performance on a real environment. Moreover, it is still possible to improve the system in the future, and some of the functionalities that should be implemented are:

- Override of the setup's actuator values remotely, by the user, in an emergency case. This can be achieved by sending a special message to the specific SAN, stating that the specific actuator value shall be the one sent;
- Introduce a selection for the periodicity of the log messages sent by the SAN, by enabling the user to select it on the system interface. The SAN would receive such information alongside the setup data, with the RTC checking when the time is due;
- Establish a two-level authentication system, with an administrator able to control the experiments, and managing the users and their roles, by giving the Users and ID which would differentiate the users, and allow for the web page to know which information to display;
- Incorporation of more sensors to the SAN, such as an heartbeat sensor;
- Send warnings through SMS, to the user's phone numbers, by adding another field, with the phone number information, to the user table, and by employing a GSM Module on the BBB.

On a higher level of difficulty, it would be interesting to develop a way for the SANs to automatically detect the presence of additional sensors connected to it, share the information with the CCN, which would then, if needed, update the tables on the RDBMS. This would allow for much more flexible setups, and also to enhance the experiments complexity.

Appendix A

User Manual

This chapter presents the indications on how to use the system, and its software, in order to ensure that it works properly

A.1 Setting Up the Hardware

In order to guarantee the system is properly working, there are some steps taken into consideration, regarding the SAN and the CCN, which are described in this section.

A.1.1 SANs

After connecting all the sensors and actuators to the SAN, the user must then power it, with a 5V adaptor, and must certify that the switch present on the shield is on "MICRO" position. Afterwards, the SAN will be ready to receive a setup and start the experiment. If its behaviour seems wrong, the user can connect the SAN to their computer, and check, in real time, what is happening by connecting to the SAN via a terminal.

Additionally, the user must have special attention to the switch present on the Wireless SD Shield, as the communications via the Xbee Modules only happens when the switch's position is *Micro*, as shown on Figure A.1.

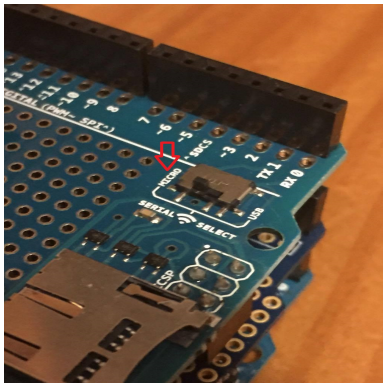


Figure A.1: Xbee Switch Position

A.1.2 CCN

To get the CCN to properly work, the user has to provide it with power, through a XV adaptor, and ethernet, or and Internet Via USB connection. Internet connection is required, as its necessary for getting the current data from an NTP server, and also for accessing the database. Furthermore, the user shall connect the Xbee Explorer USB, with the Xbee defined as coordinator connected to it, to the CCNs USB port. Considering that all these steps are correctly followed, the CCN is ready to go.

A.2 System Interface

Once the user accesses the System Interface, he we will presented with the login page, with the login box shown on Figure A.2.

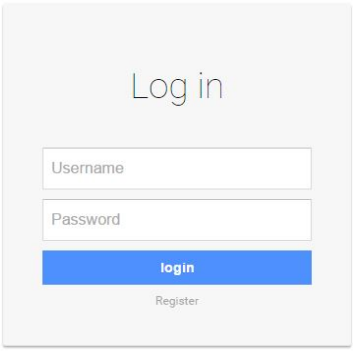


Figure A.2: Login

If the user does not have a login, he must proceed to the register page, with the register box shown on Figure A.3.

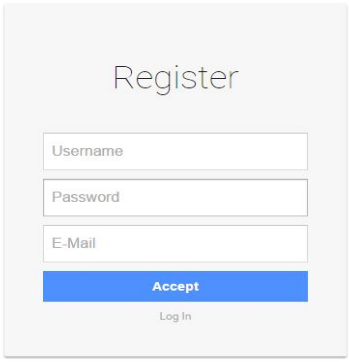


Figure A.3: Register

After the register formulary is sent, the user will return to the login page. After the login is done, the user is sent to the main page, which is shown in Figure A.4.

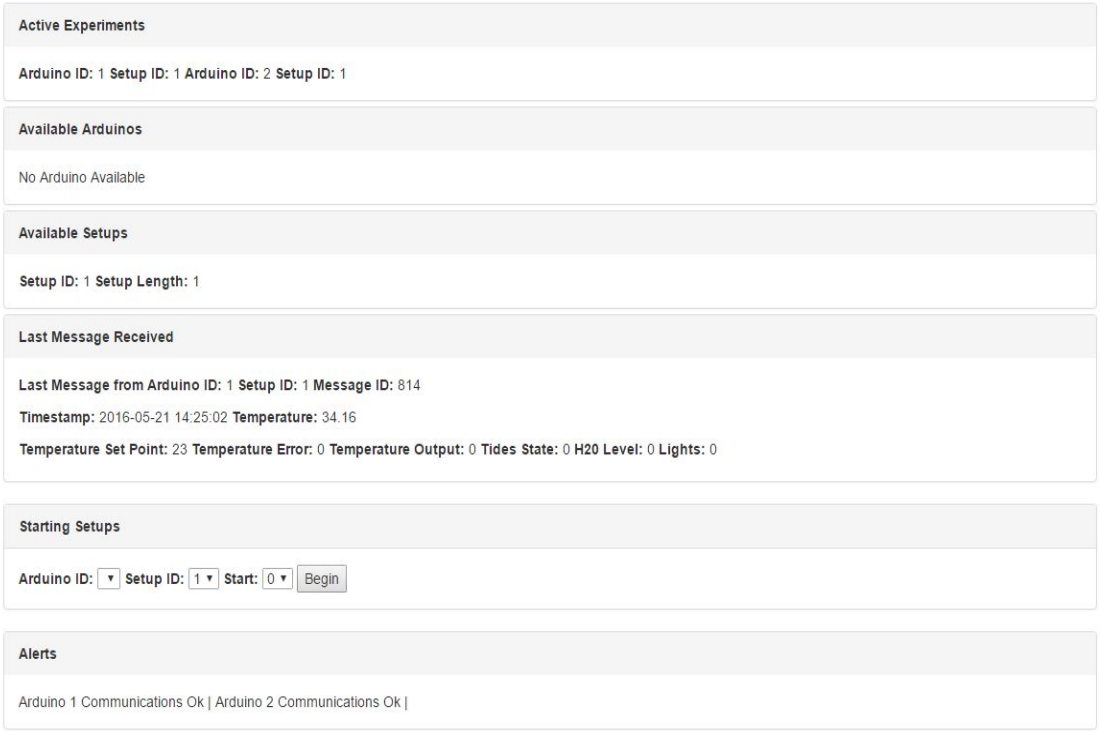


Figure A.4: Main Page

Each page has the menu shown on A.5 on the top, which the user must use to navigate throught the different pages of the website.

Figure A.5: Website Menu

If the user wants to start a new experiment, he must do the following:

- Check if there are available Setups;
- Check if there are available SANs;

Assuming that both Setup and SANs are available, the user can select which SAN he wants running a specific Setup, and change the Start value from 0 to 1. After that, the user must click on the Begin Button for the changes to take effect, which is depicted on Figure A.6

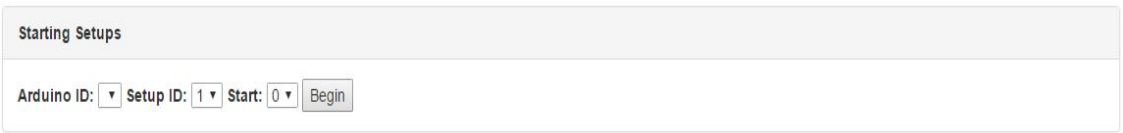


Figure A.6: Setup Selector

On the other hand, if there are no available Setups, the user shall move to the setup insert page, where he must specify the Setup ID, and Setup length, and then insert the setup data, as shown on Figure A.7. A setup will only be deemed as active if, as previously stated, its specified length and number of rows related to that specific Setup ID is the same.

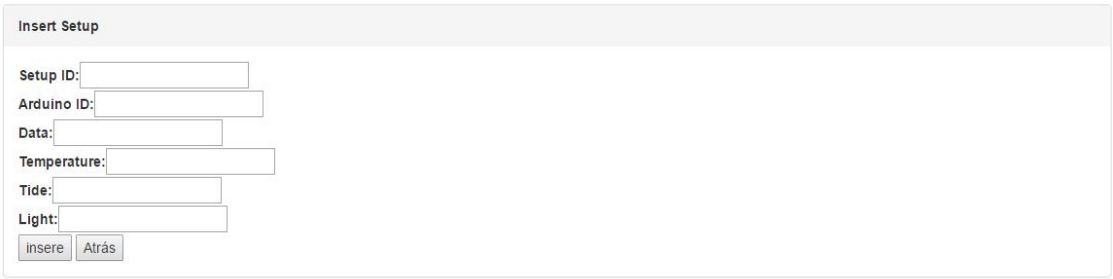


Figure A.7: Setup Questionary

Taking into consideration that on the Main Page the User can only check the Setup ID, and not the information that is carried by that ID, there is the possibility to check all the setups information, by moving the "Lista Setups" page, which is shown on Figure A.8. There the user can search through the table for the desired information, and is also able to download the data that is displayed on the screen in multiple formats, such as .xls, .csv, .pdf, or even copy it to the clipboard.

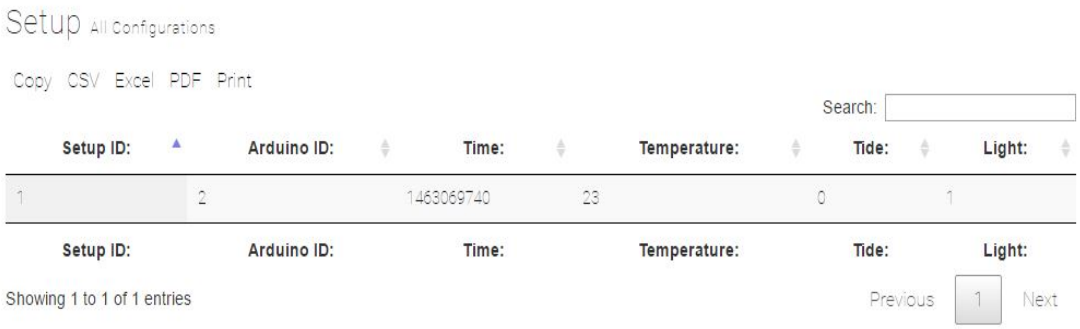
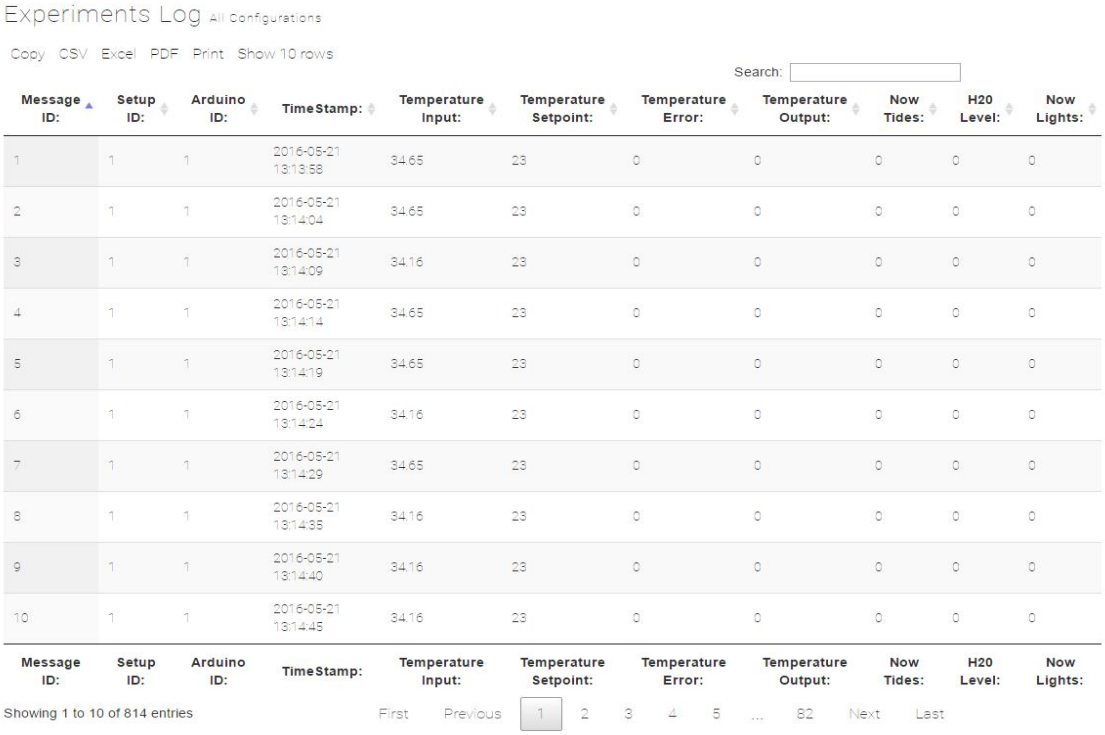


Figure A.8: Tables Presenting Setups Stored

When the user wants to see the data from the experiments, he has two options, either seeing it displayed on a table, where logging data from all the experiments is present, as shown on Figure A.9, or by moving to the graphics page, as exemplified on Figure A.10.

On the webpage displaying logged data, all the information regarding every SAN ID and Setup ID is presented in a single table. The user is able to select specific information, by using the search field. As explained beforehand, the data represented in tables can be downloaded in multiple formats, however, the file downloaded will only contain data that is presetend on the webpage, meaning that if, for example, the logged data exceeds 25 rows, the user must select the option to show 50, 100 or all rows, so that information is all stored on the downloaded file.



Experiments Log All configurations

Copy CSV Excel PDF Print Show 10 rows

Search:

Message ID:	Setup ID:	Arduino ID:	TimeStamp:	Temperature Input:	Temperature Setpoint:	Temperature Error:	Temperature Output:	Now Tides:	H20 Level:	Now Lights:
1	1	1	2016-05-21 13:13:58	34.65	23	0	0	0	0	0
2	1	1	2016-05-21 13:14:04	34.65	23	0	0	0	0	0
3	1	1	2016-05-21 13:14:09	34.16	23	0	0	0	0	0
4	1	1	2016-05-21 13:14:14	34.65	23	0	0	0	0	0
5	1	1	2016-05-21 13:14:19	34.65	23	0	0	0	0	0
6	1	1	2016-05-21 13:14:24	34.16	23	0	0	0	0	0
7	1	1	2016-05-21 13:14:29	34.65	23	0	0	0	0	0
8	1	1	2016-05-21 13:14:35	34.16	23	0	0	0	0	0
9	1	1	2016-05-21 13:14:40	34.16	23	0	0	0	0	0
10	1	1	2016-05-21 13:14:45	34.16	23	0	0	0	0	0

Showing 1 to 10 of 814 entries First Previous 1 2 3 4 5 ... 82 Next Last

Figure A.9: Tables Presenting Logged Data

To select the information represented in the graphics, the process is similar to the setup selector on the main page. The user must select the SAN ID and Setup ID, with the choice being restricted to SANs that actually are running, or ran, the selected Setup ID. Afterwards, the user has to click on the Select button, for the query to be sent. Finally, the page is refreshed and the graphics will contain the information regarding the selected options, as exemplified in Figure A.10

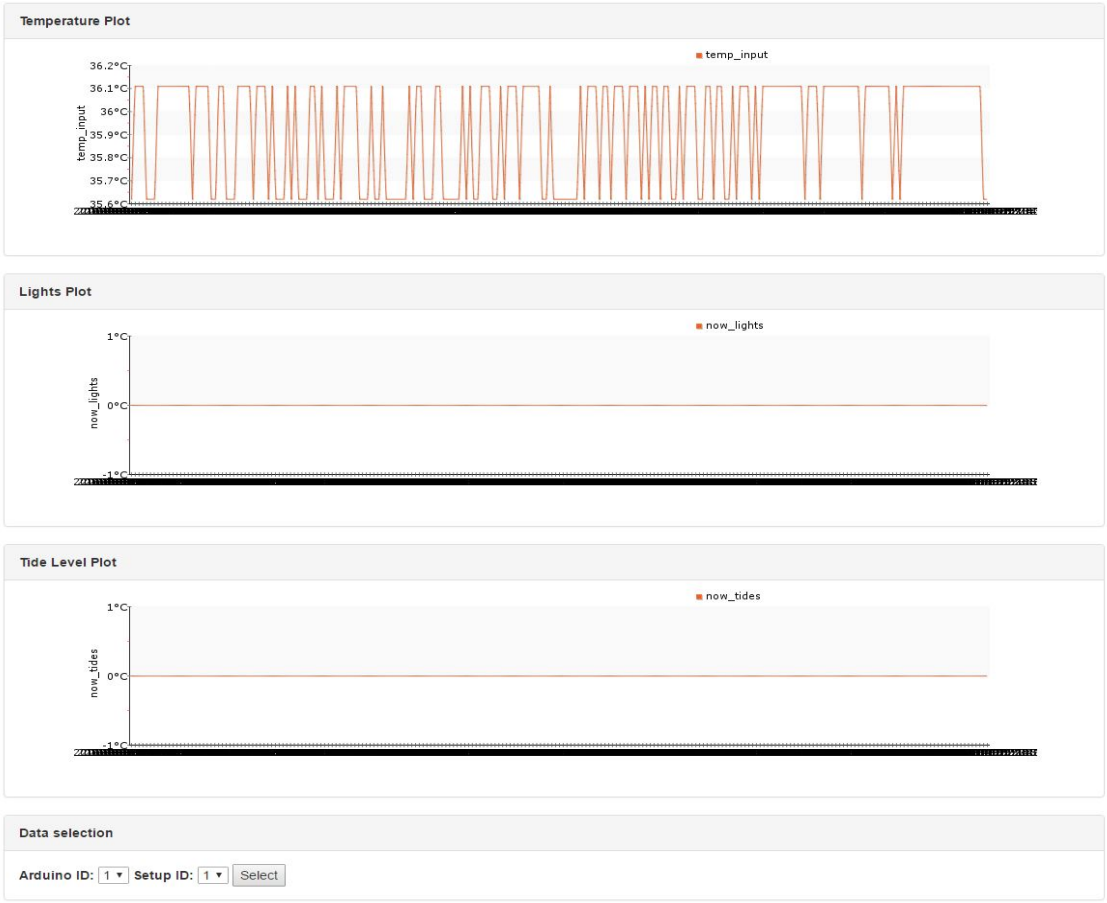


Figure A.10: Graphics Presenting Data

Appendix B

Abstract Submitted to OCEANS'16 MTS/IEEE Monterey Conference

A remote monitoring and control system for ecosystem replication experiments

João Pinto Ventura
Faculty of Engineering
University of Porto
Email: ee10253@fe.up.pt

Nuno Cruz
Faculty of Engineering and
INESCTEC
University of Porto
Email: nacruz@fe.up.pt

Fernando P. Lima
Faculty of Sciences and
CIBIO/InBIO
University of Porto
Email: fplima@gmail.com

***Abstract*—In this article we describe the implementation of remote monitoring and control for multiple and independent experiments, namely, ecosystem replication experiments, first by presenting the main concepts behind the system architecture, and ultimately its design, and secondly by discussing its implementation, which makes use of IEEE 802.15.4 Standard for Wireless Communications, a BeagleBone Black as the central coordinator for the experiments, and Arduino Mega as the monitoring and control device for each experiment.**

***Index Terms*—Networked Controlled Systems, Remote Laboratories, Wireless Sensor and Actuator Networks, Wireless Sensor Networks.**

I. Introduction

The study of ecosystems, more specifically, aquatic ones, is significant in order to determine, among others, the effects of climate change on species' distribution [1].

Aforesaid studies can be done in the ecosystem itself or through a replication of it in a laboratory. However, the study of natural ecosystems, such as the aquatic one, has been proven to be a difficult task, mainly because of the logistical issues it imposes, which are often impossible to deal with and its high costs [2]. To deal with such difficulties, enclosed experiments have been used, as they enable the replication of ecosystems in controlled and repeatable conditions, which can be duplicated and validated by fellow researches worldwide.

Since the experiments have a fluctuating time span, from weeks up to years [1], in longer experiments the necessity for remote monitoring and control is bigger, as the impact of a single event, such as unexpected deviation of water temperature, can render the experiment unsuccessful. In the last several years, sensor networks have been used to fully measure such parameters of interest however, the increasing size of the areas to be monitored has led to wireless sensor networks, which have become increasingly appealing as both costs and power consumption become lower.

The necessity for controlling environmental conditions has made the inclusion of actuators on said network a logical step, thus leading to the creation of Wireless Sensor and Actuator Networks (WSAN), as it enables the alteration of the environment by use of, for example, heating or light sources. With the incorporation of actuators, however, the complexity of the communications in the network is increased, as the protocols have to manage many-to-one communications with sensors when these are forwarding data, and one-to-many communications when the data needs to be sent to the actuators [3].

The link to an external network with the capacity to monitor and control the ongoing experiments is directly related to the concept of Remote Laboratories, which aims at providing all the functionalities that a laboratory has physically, but over the Internet, such as total control of the experiments and all its instruments, as well as gathering all the data in one place.

II. Networks for systems' monitoring and control

A. Networked Control Systems

NCS are defined by [4] as "spatially distributed systems in which the communication between sensors, actuators and controllers occurs through a shared band-limited digital communication network".

Two control systems that use network communications are defined on [5], with them being shared-controlled systems and remote control systems. On shared network connections, the sensor and actuator groups communicate with their associated controller, sharing the same network

B. Wireless Sensor and Actuator Networks

Wireless Sensor Networks (WSN) consists on a network of sensor nodes used to monitor certain environment, which then forward the data acquired to a central controller (or sink), using wireless communications, that either uses the information or relays it to other networks, such as the Internet, through the use of a gateway, thus enabling the interaction between user and the network to be made remotely. [3]

The main difference between WSN and WSAN lies on the presence of actuator nodes, which have the ability to alter the environment, either by a scheduled event or in response to an input change ([6],[3]), essentially becoming a closed-loop feedback system. In contrast to WSNs, WSANs are not limited to low power consumption devices, as some actuators such as water pumps, or motors, require substantial energy to operate.

III. Remote Laboratories

On [7], remote laboratories are defined as online environments for operating instruments and collecting measurement

data over the Internet. They are mostly used at universities for educational purposes [8], as they provide unlimited access to the experiment at all times, affording students the possibility of autonomous work and learning, whilst preventing damages to the equipment and reducing the strain on laboratory occupation time.

IV. System architecture

Using concepts associated with NCS, WSANs and Remote Laboratories, a final decision on the system architecture was made, and its design is as shown on figure 1.

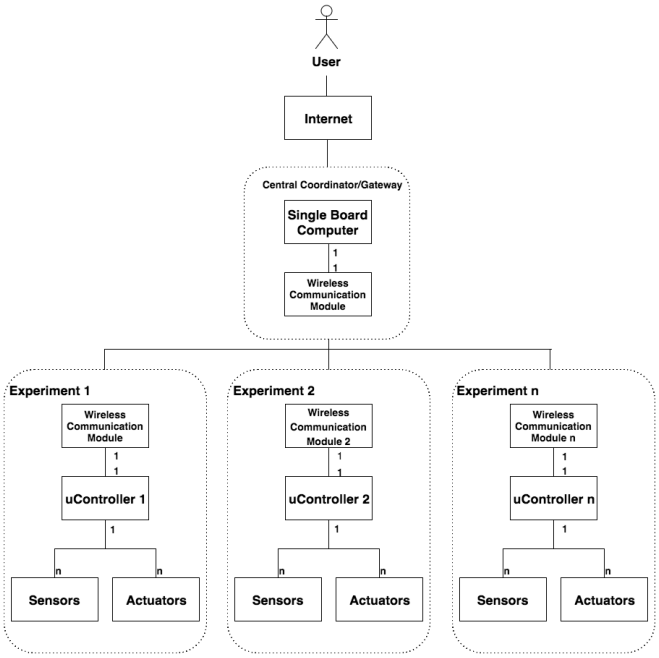


Figure 1. System Architecture

On this architecture, we have the experiment controllers and their respective sensor and actuators. The Architecture consists on a one-to-one star based topology, using a ZigBee Protocol, connecting the experiments with the controller.

V. System Overview

For the implementation of this architecture, the hardware chosen was:

- **Micro Controller:** Arduino Mega
- **Central Coordinator:** BeagleBone Black Rev C
- **Wireless Communication Module:** Xbee PRO Series 1

The Central Controller role will be performed by a BeagleBone Black. Even though the technical specifications of the Raspberry Pi 2 for this application are superior in comparison to the BeagleBone Black, there is an exterior constraint that makes this solution better suited as it might be of future interest the integration between this system and remote and as such, it was decided that hardware that could be shared between the two would be the way to go.

The microcontroller choice was restricted to the Arduino Mega since it was being previously used to control the experiments and software was already developed for it. Choosing the BeagleBone Black was due to the possibility of incorporating, in the future, other applications that are being paralelly developed for it.

The communications module was chosen as it uses the 802.15.4 standard (which is the base for Zigbee), enabling as many as 65000 nodes in the network, making it possible to monitor and control that amount of experiments.

The higher level stacks of this project concern data access and storage. The data will be stored on a PostgreSQL database and users will access it in a website, with two authentication levels:

1) Administrator, which will be shared between the head researchers, has full authority over the system, being able to upload new experiment setups, change desired values and actuator levels in real time conditions, check sensor values and alarms.
2) User which will only be able to view data and alarms, with no possibility of real-time control.

VI. Conclusion

In the final paper we will provide details of the implementation of the 802.15.4 protocol between the central coordinator and the individual experiments, and how the user can access data, and control the experiments remotely.

References

[1] M. Gandra, R. Seabra, and F. P. Lima, "A Low-Cost, Versatile Data Logging System For Ecological Applications," *Limnology and Oceanography: Methods*, vol. 13, no. 3, pp. 115–126, 2015. [Online]. Available: http://dx.doi.org/10.1002/lom3.10012

[2] D. W. Schindler, "Whole-Ecosystem Experiments: Replication Versus Realism: The Need for Ecosystem-Scale Experiments," *Ecosystems*, vol. 1, no. 4, pp. 323–334, 1998.

[3] G. M. A. C. Roberto Verdone, Davide Dardari, *Wireless Sensor and Actuator Networks Technologies*. Academic Press, 2008.

[4] J. P. Hespanha, P. Naghshtabrizi, and Y. Xu, "A Survey of Recent Results in Networked Control Systems," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 138–172, 2007.

[5] R. Gupta and M.-Y. Chow, "Overview of Networked Control Systems," *Networked Control Systems SE - 1*, pp. 1–23, 2008. [Online]. Available: http://dx.doi.org/10.1007/978-1-84800-215-9_1

[6] R. S. J. Reyes, J. C. Monje, M. E. C. Santos, L. A. Mateo, R. L. G. Espiritu, J. V. Isiderio, C. M. M. Lacson, and R. E. T. Ocfemia, "Throughput, Power and Cost Comparison of Zigbee-based and ISM-based WSAN Implementations," *Proceedings of the 8Th Wseas International Conference on Applied Electromagnetics, Wireless and Optical Communications*, pp. 61–66, 2010.

[7] A. Maiti, A. A. Kist, and A. D. Maxwell, "Real-Time Remote Access Laboratory With Distributed and Modular Design," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 6, pp. 3607–3618, 2015.

[8] P. Anzhelika, G. Olga, E. Ivanov, A. Sokolyanskii, and S. Kurson, "Development and Application of Remote Laboratory for Embedded Systems Design," *12th International Conference on Remote Engineering and Virtual Instrumentation (REV)*, vol. 11, no. 3, pp. 69–73, 2015.

References

[1] Miguel Gandra, Rui Seabra, and Fernando P Lima. A Low-Cost, Versatile Data Logging System For Ecological Applications. *Limnology and Oceanography: Methods*, 13(3):115–126, 2015. URL: http://dx.doi.org/10.1002/lom3.10012, doi:10.1002/lom3.10012.

[2] David W. Schindler. Whole-Ecosystem Experiments: Replication Versus Realism: The Need for Ecosystem-Scale Experiments. *Ecosystems*, 1(4):323–334, 1998. doi:10.1007/s100219900026.

[3] J E Petersen, V S Kennedy, W C Dennison, and W M Kemp. *Enclosed Experimental Ecosystems and Scale*. 2009. doi:10.1007/978-0-387-76767-3.

[4] Gianluca Mazzini Andrea Conti Roberto Verdone, Davide Dardari. *Wireless Sensor and Actuator Networks Technologies*. Academic Press, 2008.

[5] Joao P. Hespanha, Payam Naghshtabrizi, and Yonggang Xu. A Survey of Recent Results in Networked Control Systems. *Proceedings of the IEEE*, 95(1):138–172, 2007. doi:10.1109/JPROC.2006.887288.

[6] Rachana A. Gupta and Mo-Yuen Chow. Overview of Networked Control Systems. *Networked Control Systems SE - 1*, pages 1–23, 2008. URL: http://dx.doi.org/10.1007/978-1-84800-215-9{_}1, doi:10.1007/978-1-84800-215-9{_}1.

[7] Rosula S J Reyes, Jose Claro Monje, Marc Ericson C Santos, Lorlynn A Mateo, Roma Lynne G Espiritu, John Vianney Isiderio, Carlos Miguel M Lacson, and Ray Edwin T Ocfemia. Throughput, Power and Cost Comparison of Zigbee-based and ISM-based WSAN Implementations. *Proceedings of the 8Th Wseas International Conference on Applied Electromagnetics, Wireless and Optical Communications*, pages 61–66, 2010.

[8] Ian F. Akyildiz and Ismail H. Kasimoglu. Wireless Sensor and Actor Networks: Research Challenges. *Ad Hoc Networks*, 2(4):351–367, 2004. doi:10.1016/j.adhoc.2004.04.003.

[9] Ananda Maiti, Alexander A. Kist, and Andrew D. Maxwell. Real-Time Remote Access Laboratory With Distributed and Modular Design. *IEEE Transactions on Industrial Electronics*, 62(6):3607–3618, 2015. doi:10.1109/TIE.2014.2374572.

[10] Parkhomenko Anzhelika, Gladkova Olga, Eugene Ivanov, Aleksandr Sokolyanskii, and Sergey Kurson. Development and Application of Remote Laboratory for Embedded Systems Design. *12th International Conference on Remote Engineering and Virtual Instrumentation (REV)*, 11(3):69–73, 2015.

[11] Luís Gomes and Seta Bogosyan. Current Trends in Remote Laboratories. *Industrial Electronics, IEEE Transactions on*, 56(12):4744–4756, 2009. doi:10.1109/TIE.2009.2033293.

[12] Ignacio Angulo, Javier Garcia-Zubia, Pablo Orduna, and Olga Dziabenko. Addressing Low Cost Remote Laboratories Through Federation Protocols: Fish Tank Remote Laboratory. *IEEE Global Engineering Education Conference, EDUCON*, pages 757–762, 2013. doi:10.1109/EduCon.2013.6530192.

[13] A. Agrawal and S. Srivastava. WebLab: A Generic Architecture for Remote Laboratories. *Proceedings of the 15th International Conference on Advanced Computing and Communications (ADCOM 2007)*, pages 301–306, 2007. doi:10.1109/ADCOM.2007.71.

[14] Andreas Willig, Hagen Woesner, I P Internetworking, Lucia Lo Bello, Wolfgang Kampichler, Internet Security, and Christopher Kruegel. 3 A Perspective on Internet Routing : IP Routing Protocols and Addressing Issues. 2005.

[15] Jin-shyan Lee, Yu-wei Su, and Chung-chou Shen. A Comparative Study of Wireless Protocols. *IECON Proceedings (Industrial Electronics Conference)*, pages 46–51, 2007. doi:10.1109/IECON.2007.4460126.

[16] Jaypal Baviskar, Afshan Mulla, Amol Baviskar, Shweta Ashtekar, and Amruta Chintawar. Real Time Monitoring and Control System for Green House Based on 802.15.4 Wireless Sensor Network. *2014 Fourth International Conference on Communication Systems and Network Technologies*, (December 2004):98–103, 2014. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6821365, doi:10.1109/CSNT.2014.28.

[17] Siva V Girish, S Dhileep Shyl, R Prakash, and A Balaji Ganesh. Development of WSAN Based Unmanned Greenhouse Management System. (JANUARY 2015), 2015.

[18] Sergio Garcia Gil. Characteristics and evaluation of ieee 802.11n standard. 2013.

[19] ShamKant B. Navathe Ramez Elmasri. *Fundamentals of Database Systems*. Pearson, 2015.

[20] David J. Auer David M. Kroenke. *Database Processing: Fundamentals, Design, and Implementation*. Pearson, 2016.

[21] Geoff Moes Robert Sheldon. *Beggining MySQL*. Wiley Publishing, Inc., 2005.

[22] Frederico Tavares. *Desenvolvimento de Aplicações em PHP*. FDA - Editora de Informática, 2012.

[23] Kelly C. Bourne. *Application Administrators Handbook. Installing, Updating and Troubleshooting Software*. Morgan Kaufmann, 1 edition, 2014.

[24] phppgadmin. http://phppgadmin.sourceforge.net/doku.php. Accessed on 15th of May, 2016.

[25] Xampp web page. https://www.apachefriends.org/index.html. Accessed on 15th of May, 2016.

[26] MaxStream. *XBee ™ / XBee-PRO ™ OEM RF Modules*, 2005.

[27] Libxbee download source. https://github.com/attie/libxbee3. Accessed on 15th of May, 2016.

[28] Libxbee library man files. http://doc.libxbee.attie.co.uk/man3. Accessed on 15th of May, 2016.

[29] Postgresql 9.1 documentation. https://www.postgresql.org/docs/9.1/static/index.html. Accessed on 15th of May, 2016.

[30] Xbee arduino library. https://github.com/andrewrapp/xbee-arduino. Accessed on 15th of May, 2016.

[31] Datatables website. https://datatables.net/. Accessed on 15th of May, 2016.

[32] pchart 2.0 official web page. http://www.pchart.net/. Accessed on 15th of May, 2016.

[33] Phpmailer. https://github.com/PHPMailer/PHPMailer. Accessed on 15th of May, 2016.